МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

«Мікропроцесорні системи». Комп'ютерний практикум

*Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для студентів,
які навчаються за спеціальністю 171 «Електроніка»,
спеціалізацією «Акустичні мультимедійні системи і технології обробки музично-мовної інформації»*

Київ
КПІ ім. Ігоря Сікорського
2019

«Мікропроцесорні системи». Комп'ютерний практикум: навч. посіб. для студ. спеціальності 171 «Електроніка», спеціалізації «Акустичні мультимедійні системи і технології обробки музично-мовної інформації» / Л.М.Батрак, Д.А.Миколаєць, В.А.Тодоренко, О.В.Хоменко, ; КПІ ім. Ігоря Сікорського. – Електронні текстові данні [1 файл: 2,59 Мбайт]. – Київ : КПІ ім. Ігоря Сікорського,2019. – 94 с.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 9 від 30.05.2019р) за поданням Вченої ради факультету електроніки (протокол № 04 / 2019, від 26.04. 2019р.)

Електронне навчальне видання

«Мікропроцесорні системи».
Комп'ютерний практикум

Укладачі: *Батрак Лариса Миколаївна,,* канд. техн. наук, доц.
Миколаєць Дмитро Анатолійович, канд. техн. наук, доц.
Тодоренко Віктор Агафонович, канд. техн. наук, доц.
Хоменко Олександр Васильович
Відповідальний редактор *Ямненко Ю.С.,* завідувач кафедри промислової електроніки, д-р техн. наук, проф.
Рецензент *Михайлов С.Р.* доцент кафедри електронних приладів та пристроїв, канд. техн. наук, доц.

У посібнику наведено вимоги та методичні рекомендації до виконання комп'ютерного практикуму з кредитного модуля «Мікропроцесорні системи». Розглянуті загальні положення, де визначено актуальність виконання робіт та їх відповідність лекціям кредитного модуля.

До кожного лабораторного практикуму визначено мету роботи, перелік завдань, програму роботи, вимоги до оформлення звіту, контрольні запитання. Кожна робота супроводжується теоретичними відомостями. В методичних рекомендація приведено перелік джерел інформації, які доцільно використовувати при самостійній роботі по підготовці до лабораторних практикумів.

Видання призначено для студентів технічних вузів, які навчаються за спеціальністю спеціальності 171 «Електроніка», спеціалізації «Акустичні мультимедійні системи і технології обробки музично-мовної інформації». Може бути використано студентами інших спеціальностей, при вивченні дисципліни «Мікропроцесорні системи». Теоретичні відомості до лабораторних робіт та контрольні запитання можуть також використовуватись для підготовки до підсумкової атестації з навчальної дисципліни.

© КПІ ім. Ігоря Сікорського, 2019

ЗМІСТ

ВСТУП

У методичних вказівках розглядаються питання розробки та відпрацювання програмного забезпечення мікроконтролерних систем побудованих на AVR - мікроконтролерах фірми Atmel з використанням мови Сі.

Метою комп'ютерного практикуму є вивчення основних принципів побудови мікропроцесорних пристроїв управління та обробки інформації, набуття практичних навичок побудови апаратурної частини та програмного забезпечення та методів їх налагодження.

Під час виконання практикуму студенти набувають знань про методи побудови мікропроцесорних пристроїв управління об`єктами промислової електроніки, систем збору та обробки інформації, основні методи налагодження програмного забезпечення та налаштування апаратурної частини.

Підготовлені методичні вказівки до шести робіт. Кожна робота містить мету практикуму, порядок виконання, індивідуальні завдання, перелік контрольних запитань, перелік літератури та теоретичні відомості. У роботах наведені приклади виконання елементів практикуму та наскрізного проекту.

Підготовка до кожного практикуму включає детальне ознайомлення з теоретичними відомостями, що містяться в роботах.

Працездатність розроблених програм перевіряється шляхом програмування мікроконтролера спеціалізованого стенду та дослідження реакцій його зовнішніх пристроїв.

1. КОМП'ЮТЕРНИЙ ПРАКТИКУМ №1.
РОБОТА З ПАМ'ЯТТЮ AVR-МІКРОКОНТРОЛЕРА

1. Мета роботи:

- ознайомлення з інтегрованим середовищем розробки прикладного програмного забезпечення AVR мікроконтролерів – Image Craft (ICC);
- ознайомлення з середовищем програмування Pony Prog;
- розробка пробного проекту по роботі з різними сегментами пам'яті AVR мікроконтролерів;
- програмування мікроконтролера.

2. Програма роботи:

2.1 Ознайомитися з інтегрованим середовищем розробки та відпрацювання прикладного програмного забезпечення ICC та програматором Pony Prog;

2.2 Ознайомитися з функціями та регістрами процесора, що дозволяють працювати з різними сегментами пам'яті мікроконтролера;

2.3 Створити пробний проект;

2.4 Провести компіляцію та вилучити виявлені помилки;

2.5 Провести покрокове відпрацювати програми з використанням дебагера AVR Studio;

2.6 Запрограмувати мікроконтролер. ***Увага! Перше програмування провести у присутності викладача***;

2.7 Зробити висновки по роботі.

3. Завдання до проекту лабораторної роботи

- Провести ініціалізацію констант X1, X2, X3, які розміщуються у різних сегментах пам'яті мікроконтролера (ROM, SRAM, EEPROM) за різними адресами.
- Провести задану у завданні математичну обробку.
- У залежності від результату обробки (результат обробки ">0" або "=0"), вивести інформацію на позначені у завданні пристрої стенду.
- Варіанти завдань до лабораторної роботи наведено у таблиці 1.1.

Таблиця 1.1

№	X1	X2	X3	Y		
	Сегмент, адреса	Сегмент, адреса	Сегмент, адреса	Операції	Реакція	
					"> 0"	"= 0"
1	ROM, $100	SRAM, $0007	EEPROM, 5	X1+X2∨X3	LED1	LED2
2	ROM, 200	EEPROM, 7	SRAM, $0033	X1x(X2/X3)	BEEP	LED1
3	SRAM, $0050	EEPROM, 55	ROM, $400	X1/X2-X3	LED1	LIGHT
4	SRAM, $001C	ROM, 70	EEPROM, 16	X1⊕(X2+X3)	LIGHT	BEEP
5	EEPROM, $1F	ROM, $A00	SRAM, $005E	(X1∨X2)xX3	LED2	LIGHT
6	EEPROM, 2	SRAM, $0065	ROM, $100	X1+X2/X3	BEEP	LED1

7	ROM, $100	SRAM, $0010	EEPROM, 35	X1∨X2-X3	LIGHT	LED2
8	ROM, $500	EEPROM, 1	SRAM, $005D	X1+(X2⊕X3)	LED2	LIGHT
9	SRAM, $0070	EEPROM, 4	ROM, $200	X1∨(X2xX3)	LIGHT	LED1
10	SRAM, $0001	ROM, $200	EEPROM, 15	X1x(X2-X3)	LED2	LED1
11	EEPROM, 2	ROM, $200	SRAM, $0033	X1+X2⊕X3	BEEP	LIGHT
12	EEPROM, 6	SRAM, $0100	ROM, $200	(X1+X2)xX3	LIGHT	LED2

Де:

- LED1, LED2 – світлодіоди стенду. Адреси портів, відповідно, PORTA.6, PORTA.7 (L-активні);
- BEEP – пристрій звукової індикації стенду. Адреса порту – PORTA.4 (H-активний);
- LIGHT – світлодіод підсвічування LCD-індикатора. Адреса порту – PORTA.5 (H-активний).

4. Зміст звіту

4.1 Титульний листок із назвою роботи та переліком виконавців.

4.2 Мета роботи.

4.3 Проект роботи.

4.4 Висновки.

5. Контрольні запитання

5.1 Яка послідовність дій по створенню проекту та відпрацюванню помилок у середовищі ICC?

5.2 Як використовується AVR Studio для програмної симуляції створеного проекту?

5.3 Яка послідовність дій по програмуванню мікроконтролера з використанням програми Pony Prog?

5.4 Яка організація пам’яті мікроконтролерів AVR сімейства Mega?

5.5 Визначити призначення пам’яті програм та її побудову.

5.6 Визначити призначення EEPROM пам’яті даних та її побудову.

5.7 Яка організація пам’яті даних? Визначити організацію адресного простору пам’яті даних.

5.8 Назвіть типи змінних та їх модифікаторів.

5.9 Як проводиться ініціалізація змінних в SRAM області?

5.10 Як проводиться ініціалізація даних в області пам’яті програм?

5.11 Як проводиться ініціалізація даних в EEPROM пам’яті даних?

6. Література

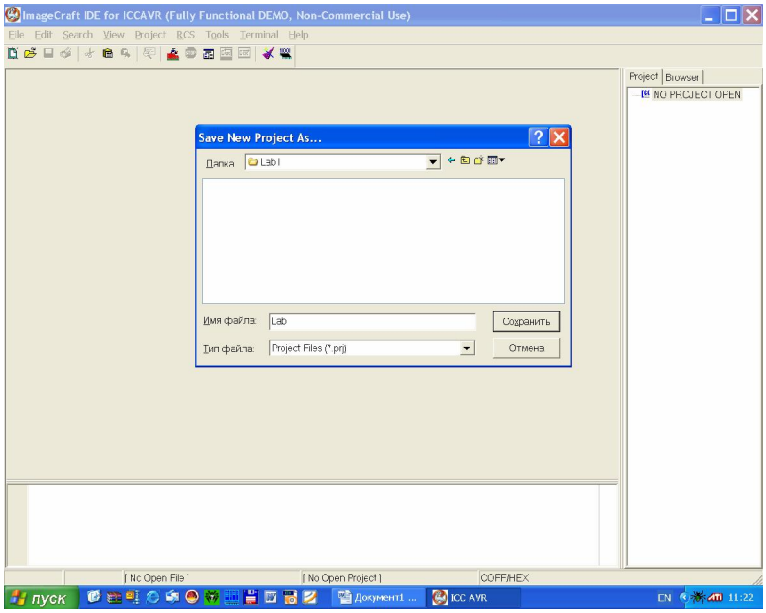
6.1 Евстифеев А.В. Микроконтроллеры AVR семейства Classic фирмы “Atmel”. – М.: Издательский дом «Додэка - XXI», 2002.

6.2 Евстифеев А.В. Микроконтроллеры AVR семейства Tiny и Mega фирмы “Atmel”. – М.: Издательский дом «Додэка - XXI», 2004.

7. Теоретичні відомості

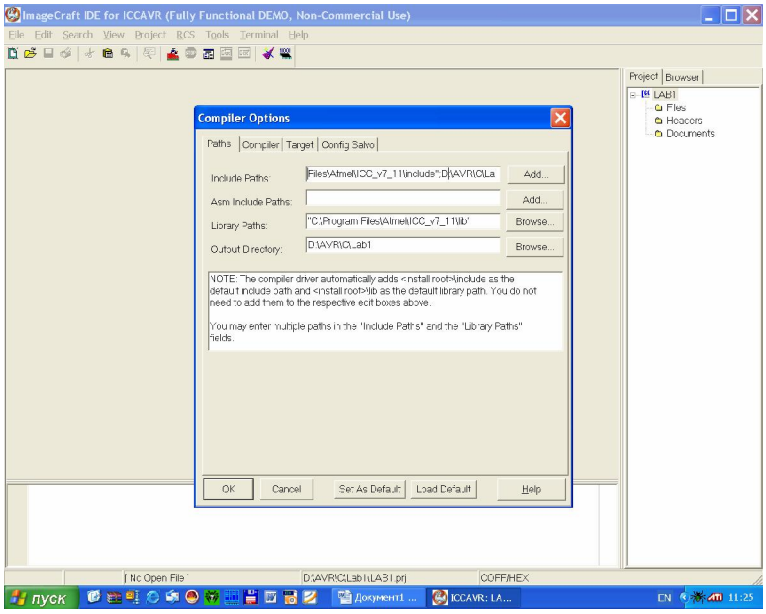
7.1 **Послідовність дій для створення проекту у середовищі ICC**

Для розробки проекту у середовищі ICC можна використовувати наступну послідовність операцій.

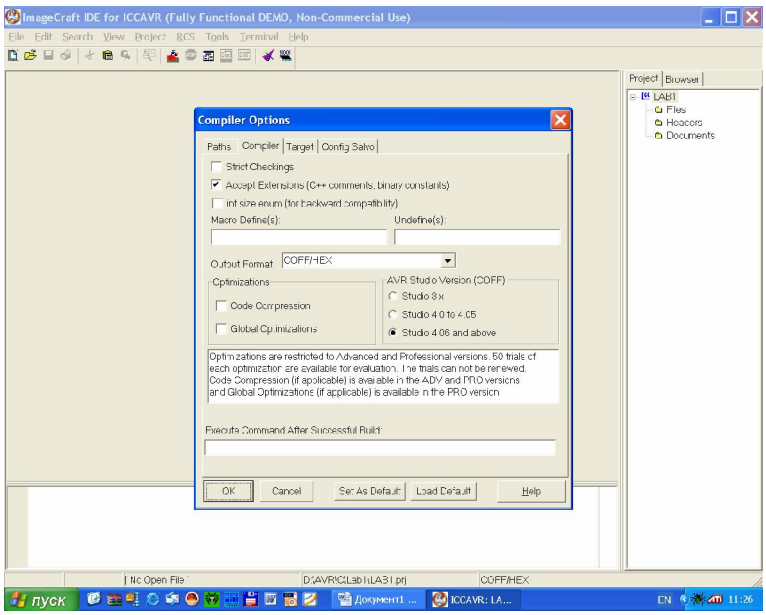


7.1.1. Кожен новий проект повинен розміщуватися у власній папці.

Виберіть у верхній стрічці меню розділ **Project** і опцію створення нового проекту **New**. На екрані монітору з'явиться вікно **Save new project as**, що використовується для запису файлу проекту. Визначте розміщення папки проекту, у разі необхідності створіть нову папку. У вікні **Имя файла** визначте ім'я Вашого проекту, а у вікні **Тип файла** оберіть розширення, що відповідає файлам проекту - .prj.



Для подальшого встановлення атрибутів проекту необхідно перейти у розділі меню **Project** до опції налаштування функцій компілятора **Options**.



У закладці **Path** визначте шляхи:

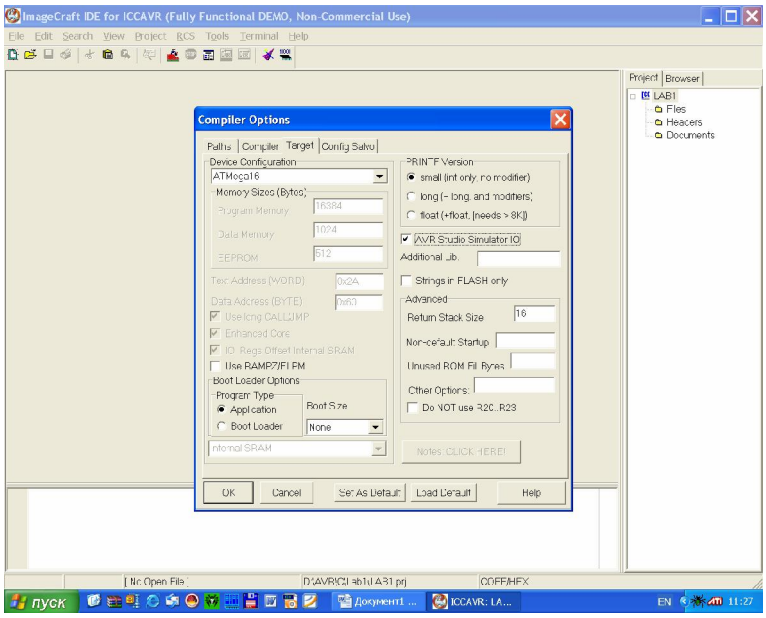
- до вкладених файлів (Include Path) С-компілятора. У цьому вікні визначте також шлях до Ваших власних вкладених файлів (папка проекту);
- до бібліотек Ci-компілятора (Library Path);
- до папки проекту (Output Directory).

У закладці **Compiler** визначте:

- необхідність використання розширень характерних для версії мови C++. Можливість використання бінарного формату констант (наприклад, 0b00110011);
- формати вихідних файлів COFF/HEX. Файл з розширенням *.cof використовується для програмної симуляції створеної програми у програмному середовищі AVRStudio. Файл з розширенням *.hex придатний для програмування мікроконтролера;
- вкажіть версію програми AVRStudio 4.06 and above;
- у разі необхідності встановіть опції оптимізації вихідних кодів (Code Compression). Ця опція звичайно використовується для програм значної ємності, коли не вистачає об'єму пам'яті мікроконтролера для розміщення прикладної програми.

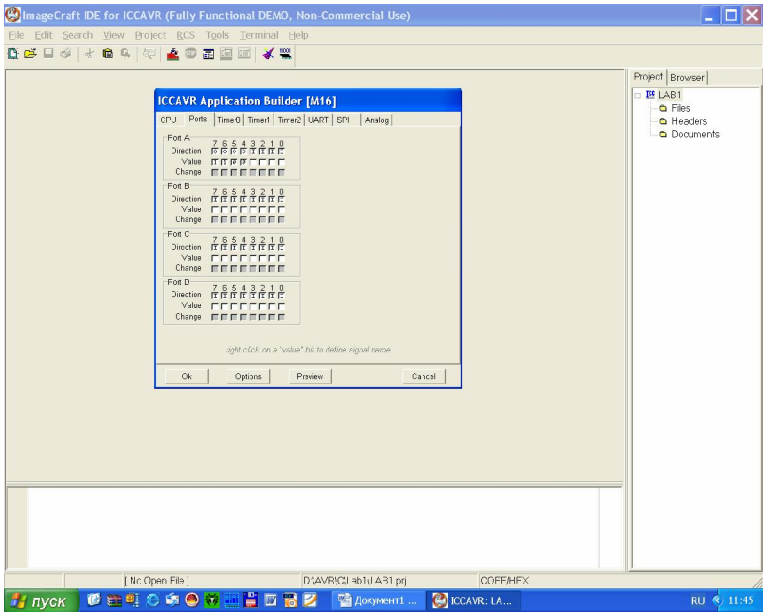
У закладці **Target**:

- у розділі Device Configuration оберіть тип мікроконтролера – ATMega16;
- у розділі Program Type визначте тип програми що створюється – прикладна програма Application;
- у розділі Boot Loader Option забороніть визначення boot- сектору у пам'яті програм – None;
- у розділі PRINTF Version встановіть позначку small;



- позначте можливість використання програми AVRStudio для програмної симуляції Вашої прикладної програми;

- якщо передбачається можливість використання фрагментів програми створених мовою асемблера, бажано для них зарезервувати файлові регістри R20…R23. У цьому разі поставте мітку Do NOT use R20…R23.

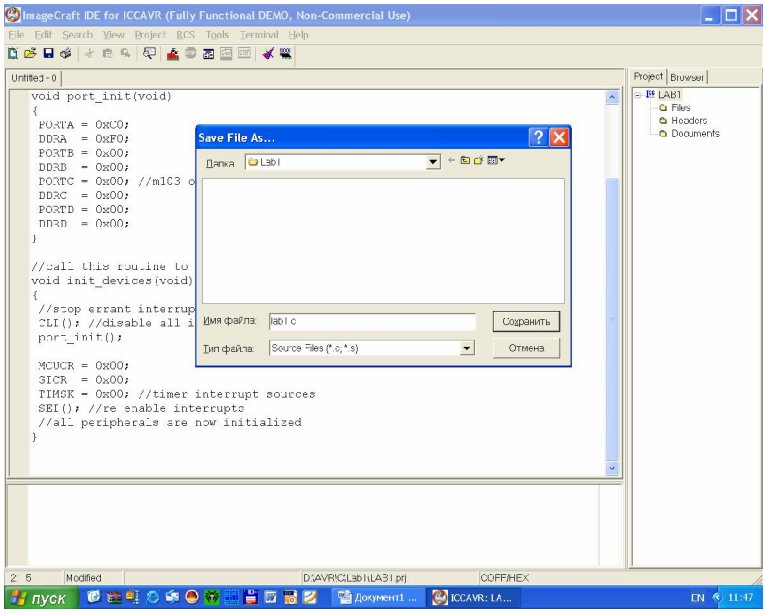


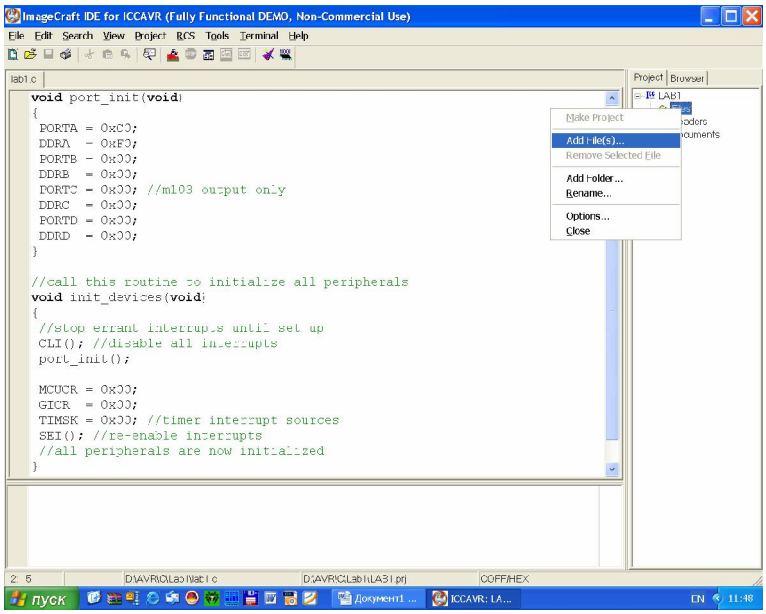
7.1.2. На наступному етапі створюється файл що компілюється. У цьому файлі визначається функція ініціалізації мікроконтролера. Для створення такого файлу у розділі меню **Tools** необхідно обрати опцію **Application Builder**.

У закладці **CPU** оберіть тип мікроконтролера ATMega16 та частоту генератора 10 МГц. Для того щоб внести у файл main-функцію необхідно за допомогою кнопки Option викликати список додаткових опцій. Оберіть у цьому списку закладку Include “main()”.

У закладці **Port** оберіть напрямки роботи ліній (приймач / передавач – I/O), та вихідний стан передавачів – “1”/”0”.

У решті закладок можна визначити початкові режими роботи таймерів 0-2, UART-контролера, SPI-контролера та аналогово-цифрового перетворювача.





Створений файл необхідно записати у папку проекту із розширенням *.c.

7.1.3. Цей файл необхідно визначити, як файл що компілюється. Для цього у розділі меню **Project** з використанням підрозділу **Add File(s)** обирають визначений файл.

7.1.4. Після встановлення всіх атрибутів проект слід записати. Ця процедура відбувається з використанням розділу верхньої стрічки меню **File** та опції запису проекту **Save**. Автоматичний запис відбувається також під час проведення компіляції.

7.1.5. Для компіляції проекту користуються кнопкою прискореного виклику **Build Project**, або у розділі меню **Project** викликається опція **Rebuild All**.

Після вдалої компіляції у вікні повідомлень з`являється текст з інформацією про використаний об’єм сегменту пам’яті програм.

На цьому етапі у папці проекту окрім основного файлу(*.c) автоматично розміщується багато файлів, серед яких виділяються:

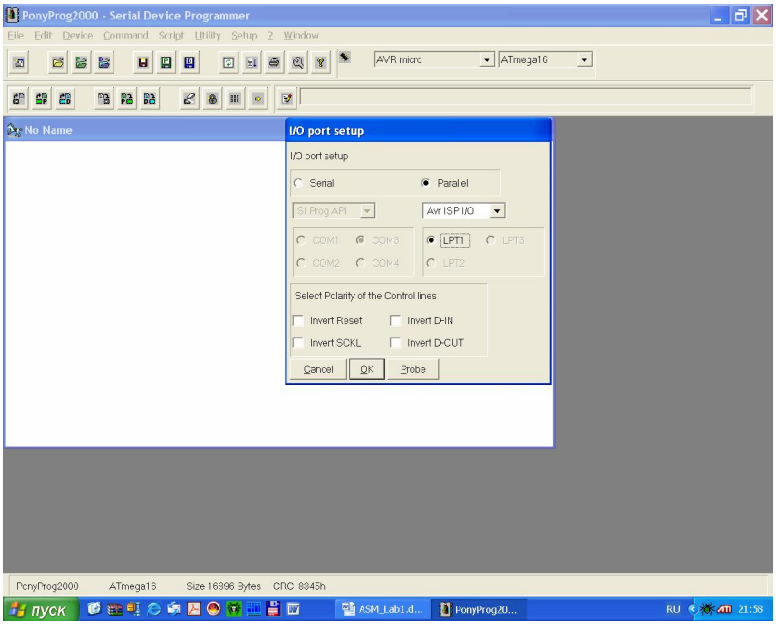
- проекту (*.prj);
- вихідний файл програмування пам’яті програм (*.hex);
- вихідний файл програмування енергонезалежної пам’яті даних (*.eep);
- лістингу (*.lis);
- об’єктний (*.o);
- зв’язку з AVRStudio (*.cof). Використовується для симулятивного налагодження програми.

7.1.6. Після невдалої компіляції програми у вікні повідомлень записуються всі виявленні помилки. Помилки позначаються червоною крапкою та вказується їх тип. Якщо рядок в якому записана знайдена помилка активізувати за допомогою подвійного натискання лівої кнопки миші, то вікно файлу з вказаною помилкою стане активним та курсор переміститься до рядка, що містить помилку. Якщо ж файл з помилкою не відкритий , то він автоматично відкриється.

7.1.7. Після закінчення роботи проект записують та закривають. Для цього у розділі меню **Project** викликають опцію **Close**.

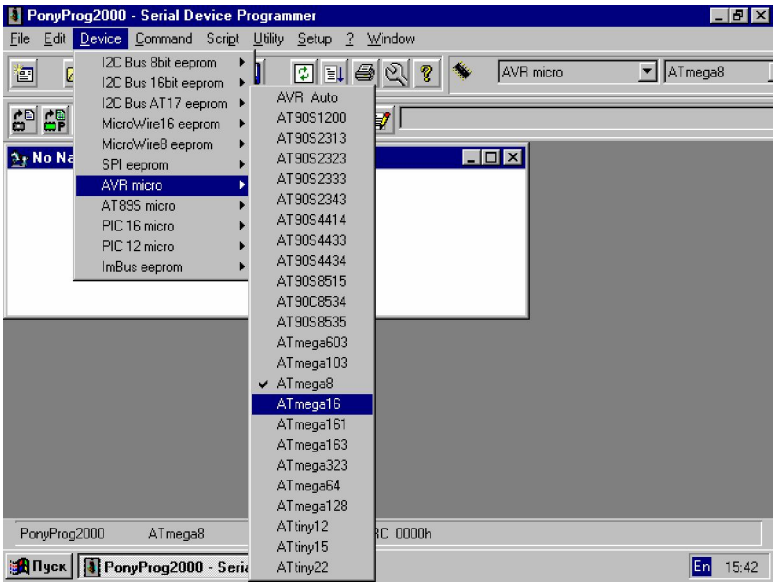
7.2. Робота із програматором Pony Prog

7.2.1. На попередньому етапі необхідно визначити тип інтерфейсу та провести калібровку програмного забезпечення програматора. У верхній стрічці меню виберіть розділ **Setup**. Для обміну з програматором оберіть



паралельний порт LPT1 та тип алгоритму обміну **AVR ISP I/O**. Виконайте перевірку каналу обміну **Probe**. Для калібровки часових характеристик програматора у розділі меню **Setup** оберіть опцію **Calibration**.

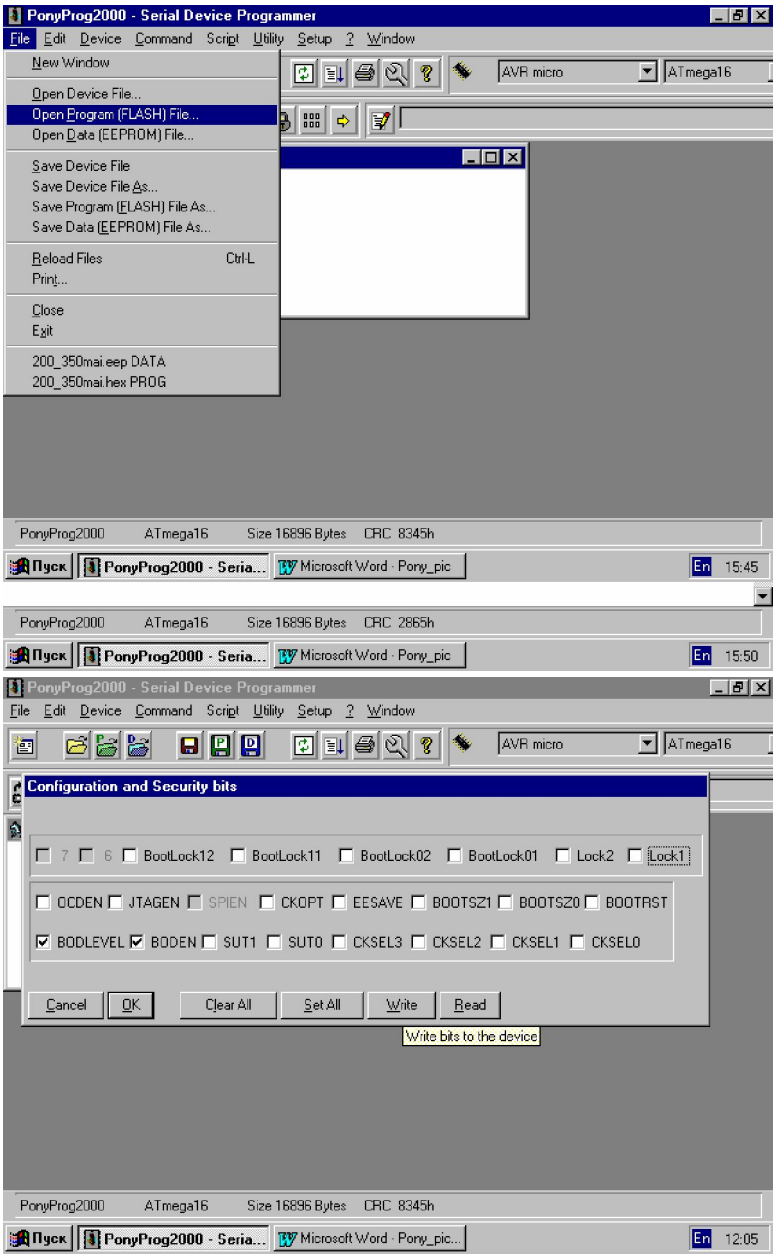
7.2.2. Необхідно визначити тип мікроконтролера. У верхній стрічці меню оберіть розділ **Device**. У випадаючих списках оберіть групу приладів **AVR micro** та мікроконтролер **ATmega16**.



7.2.3. Після цього необхідно завантажити файли для програмування сегментів пам`яті програм (Flash) та енергонезалежної пам`яті даних (EEPROM).

Для завантаження файлу програмування Flash у розділі меню **File** виберіть опцію **Open Program (Flash) File.** Потім у вікні **Open program (Flash) content file** виберіть директорію, де знаходиться Ваш проект та відкрийте файл з розширенням *.hex.

Для завантаження файлу програмування EEPROM у розділі меню **File** оберіть опцію **Open Date (EEPROM) File** та у вікні, що з'явиться після цього відкрийте файл проекту з розширенням *.eep.



У центральному вікні з використанням різних кольорів послідовно зображуються коди ROM та EEPROM сегментів пам`яті мікроконтролера.

7.2.4. Встановіть конфігурацію мікроконтролера. Для цього у розділі меню **Command** виберіть опцію **Security and Configuration Bits** (установка біта конфігурації та секретності). За допомогою команди **Read** необхідно прочитати установки мікроконтролера. У вікні, що з'явиться, необхідно визначити такі фюзеси (поставити мітки):

✓ дозвіл роботи супервізора напруги (BODEN);

✓ установка рівня напруги супервізора 4В (BODLEVEL);

✓ CKOPT – модифікації вихідної напруги тактового генератора мікроконтролера.

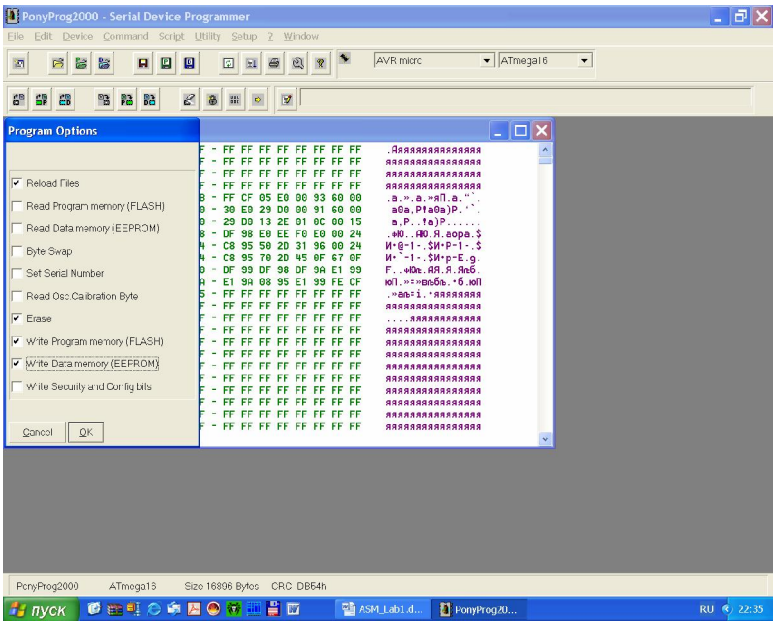
Мітки BODEN та BODLEVEL встановлюються, якщо в роботі використовуватиметься енергонезалежна пам'ять даних (EEPROM), або можливі значні коливання напруги живлення мікроконтролера.

Після чого необхідно здійснити запис вибраної конфігурації – натиснути клавішу **Write** та **Ok**.

Увага! За винятком Lock1, Lock2, забороняється встановлення інших фюзесів налаштування мікроконтролера.

7.2.5. Для спрощення процесу програмування задайте опції автоматичного програмування. Для цього в меню **Command** виберіть опцію **Program Options,** після чого у вікні що з'явиться позначте мітками наступні рядки:

✓ знищення вмісту пам’яті мікроконтролера (Erase);



✓ перезапис вихідних файлів (Reload Files);

✓ запис у пам’ять програм (Write Program memory (FLASH));

✓ запис в пам’ять даних (Write Date memory (EEPROM)).

7.2.6. Завершальний крок – автоматичне програмування мікроконтролера. Для цього в меню **Command** виберіть опцію **Program.** Також можливе використання відповідної кнопки прискореного доступу.

7.3. Типи змінних

Всі змінні та константи, що використовуються у програмі, необхідно декларувати до їх використання. Загальна форма декларації має наступний вигляд:

```
[<storage class>] typename name,
```

де: `storage class` – модифікатор доступу до змінної, `typename` – опис типу, `name` – ім’я змінної.

У таблиці 1.2 наведено основні типи змінних, їх описи та діапазони можливих значень.

Таблиця 1.2

	Тип	Варіанти	Опис	Діапазон значень
Цілі	char	беззнакові	unsigned char	0 - 255
		знакові	signed char	-128 - +127
	short	беззнакові	unsigned short	0 – 65 535
		знакові	signed short	-32 768 - +32 767
	int	беззнакові	unsigned int	0 - 65535
		знакові	signed int	-32768 - +32 767
	long	беззнакові	unsigned long	0 – 4 294 967 295
		знакові	signed long	-2 147 483 648 - + 2 147 483 647
Плаваючі	float	беззнакові	float	$3.4*10^{-38}$ - $3.4*10^{38}$
	double	беззнакові	double	$1.7*10^{-308}$ - $1.7*10^{308}$
	long double	беззнакові	long double	$3.4*10^{-4932}$ - $3.4*10^{4932}$

Змінні можуть визначатися спільно з модифікаторами, що дозволяє надавати їм певні властивості. До переліку найбільш вживаних модифікаторів

визначень змінних входять наступні ключові слова: const, volatile, static, register, extern.

Змінні можуть бути об’явлені як на зовнішньому так і внутрішньому рівнях. Якщо описи зроблено за межами усіх функцій, то об’ява вважається глобальною, або зовнішньою.

У разі визначення глобальної змінної використовують модифікатор extern. Така змінна може бути викликана з будь якої точки програми.

Визначення всередині будь якого вкладеного блоку вважається локальним, або внутрішнім. Час життя змінних, що визначаються на внутрішньому рівні розповсюджується на час виконання функції. Проте можливо модифікувати час життя такої змінної до рівня глобального, за рахунок використання модифікатору static.

У разі використання модифікатора const забороняється зміна значення змінної. Значення таких змінних (констант) записуються у пам'ять програм.

Модифікатор volatile має протилежну дію. Він вказує на те, що значення змінної може бути змінено будь якою функцією чи основною програмою.

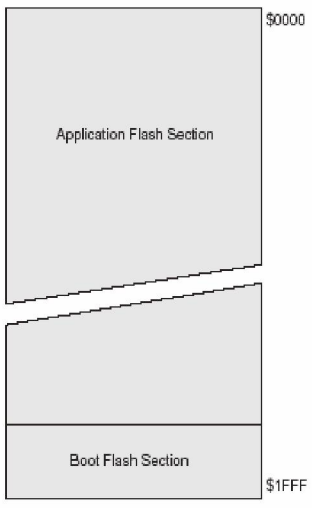
Якщо змінна часто вживається, то для зменшення об’єму програми доцільно використовувати модифікатор register. У цьому разі компілятор резервує для змінної один з вільних файлових регістрів.

Усі функції мають глобальний час життя та визначаються на зовнішньому рівні. Не допускається визначення функції всередині іншої функції.

7.4. Організація пам’яті

Пам’ять мікроконтролерів AVR організована за Гарвардською архітектурою, в якій розділені адресні простори пам’яті програм та пам’яті даних, а також і шини доступу до них.

7.4.1 **Пам’ять програм**



Пам’ять програм (FLASH) призначена для зберігання команд, що керують роботою мікроконтролера. В пам’яті програм зберігаються також різні константи, що не змінюються під час роботи програми.

Пам’ять програм виконана на базі пам’яті флеш-типу з електричним стиранням інформації. Пам’ять програм має 16-розрядну організацію. Типова довжина команди складає 16 біт (деяких – 32 біта). Для адресації пам’яті програм використовується лічильник команд (PC – Program Counter). Його розрядність відповідає об’єму пам’яті програм, для мікроконтролера Atmega16 складає 13 біт.

Пам'ять програм розділена на дві частини – сектор програми завантаження (Boot), та сектор прикладної програми. Перерозподіл розмірів секторів відбувається під час програмування, за рахунок встановлення фюзесів BOOTSZ0, BOOTSZ1.

За адресою $000 пам’яті програм знаходиться вектор скидання. Після ініціалізації (скидання) мікроконтролера виконання програми починається з цієї адреси. Існує можливість переносу адреси вектора скидання на початок Boot-сектора (встановлюється фюзес програмування BOOTRST). Для мікроконтролерів з об’ємом пам’яті програм що перевищує 8 Кбайт, за цією адресою розміщують команду переходу *JMP* до частини ініціалізації програми). Починаючи з адреси $002 у мікроконтролера ATmega16 розташовується таблиця векторів переривань.

Пам’ять програм, що використовується в мікроконтролерах AVR, розрахована не менш як на 10000 циклів стирання/запису.

Використовують наступні способи ініціалізації констант у пам`яті програм.

Якщо адреса розміщення неважлива, то застосовують варіанти, що наведені у першому прикладі.

Приклад 1

```
const unsigned int  Con1  = 1;
const unsigned char Con_Mas1[] = {1,2,3,4,5,6,7,8,9,10};
```

Якщо необхідно жорстко визначити адресу, то використовують варіант опису, який описано у другому прикладі. Недоліком такого варіанту є можливість перетину фрагментів основної програми з кодами команд та описом констант. У випадку такої події у вікні повідомлень відсутня інформація про помилку. У пам’яті програм розміщуються ті коди, які визначені останніми.

Приклад 2

```
#pragma abs_address:0x1B0
const char Titul[] = "Designed by _______09.2006";
#pragma abs_address:0x1E0
const char Name[] = "Serial number";
#pragma abs_address:0x1F0
const unsigned int Serial_number =0x0000;
#pragma end_abs_address
```

7.4.2 **Пам’ять даних**

Пам’ять даних складається з двох областей:

- статичний запам’ятовуючий пристрій (SRAM);
- енергонезалежна пам’ять даних на основі EEPROM.

Кожна з цих областей розміщена в своєму адресному просторі.

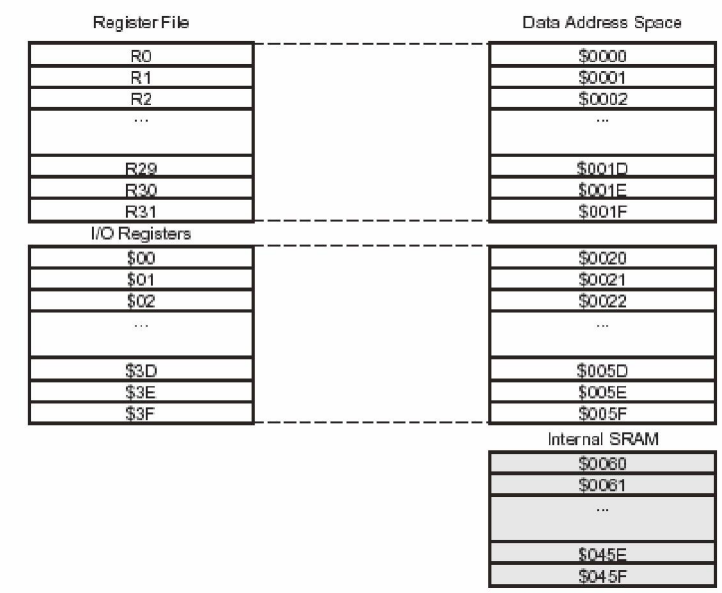
У складі SRAM області виділяють наступні сегменти:

- файловий регістр (R0 – R31);

- регістри вводу/виводу (РВВ) 64 - 128, у залежності від моделі мікроконтролера;
- регістри оперативного запам'ятовуючого пристрою (ОЗП).

Файловий регістр включає 32 регістра загального призначення, що мають символічні імена R0 – R31.

В області регістрів вводу/виводу розташовані службові регістри, а також регістри керування пристроями мікроконтролера.



Розміри області РВВ залежать від типу мікроконтролера та складають від 64 до 128 байт. Для мікроконтролера Atmega16 РВВ має об'єм у 64 байти.

Для зберігання змінних разом з регістрами файлового регістру також можуть використовуватися регістри ОЗП. Цей сегмент пам'яті використовується також для створення стекової області.

Для зберігання даних які можуть змінюватися в процесі наладки та функціонування готової системи може використовуватися енергонезалежна пам'ять даних - EEPROM. Ця пам'ять розташована в окремому адресному просторі. Доступ до неї здійснюється за допомогою спеціальних РВВ.

Регістри загального призначення. Внутрішня SRAM область

Регістри загального призначення та SRAM області можуть використовуватися для зберігання змінних.

Для резервування місця для змінної необхідно вказати її тип та ім'я.

Ініціалізація дистрибутивного значення змінної відбувається після присвоєння їй певного значення. Якщо операція ініціалізації у програмі відсутня, то змінній автоматично присвоюється нулеве значення.

Приклад 3

```
unsigned char  var1;     //резервування місця для змінної
unsigned int   var2=9;   //резервування місця та ініціалізація
```

Місце розташування змінної у файловому регістрі чи SRAM- сегменті залежить від її типу модифікаторів доступу до змінної (register, volatile, static).

Якщо при визначенні змінної тип не вказано, або використано модифікатор register, то резервується місце у сегменті файлового регістру.

У разі застосування модифікатору volatile використовуються файлові регістри R0/R1/R2/R3/R4/R5/R6/R7/R8/R9/R24/R25/R26/R27/R30/R31. Зміст

такої змінної може змінюватися у функції, що викликається. Така змінна не запам`ятовується та не поновлюється. Розташована змінна у SRAM-сегменті.

Якщо використано тип static, то початкова величина змінної запам`ятовується в файлових регістрах та копіюється у регістри SRAM-сегменту.

Регістри вводу/виводу

У мікроконтролері Atmega16 регістри вводу/виводу розташовуються у просторі розміром 64 байта.

Для обміну інформацією з регістрами вводу виводу використовують способи, що наведені у четвертому прикладі.

Приклад 4

```
unsigned char TMP;  //визначення змінної TMP
PORTA=4;            //запис у регістр PORTA константи «4»
TMP=SREG;           //читання значення регістру SREG
```

7.4.3 Енергонезалежна пам'ять даних

Пам'ять EEPROM розташована в своєму адресному просторі та має лінійну організацію.

Для звертання до змінних, що розташовані у EEPROM області, можна використовувати бібліотечні функції та макроси, або необхідно розробити власні функції.

Якщо використовуються бібліотечні функції, то для їх виклику необхідно підключити до проекту вкладений файл "eeprom.h". Для запису/читання змінних int-типу використовують функції:

- EEPROM_WRITE(int location, object) – запис двох байт.

Наприклад,

```
int i;
EEPROM_WRITE(0x1, i);    // запис  змінної "i"за адресою 0x1
```

- EEPROM_READ(int location, object) – читання двох байт.

Наприклад,

```
int i;
EEPROM_READ(0x1, i);     // читання 2 байт у змінну "i"
```

Для запису/читання змінних char-типу використовують внутрішні функції:

- unsigned char EEPROMread(int location) – читання байту з EEPROM за адресою location;
- int EEPROMwrite(int location, unsigned char byte) – запис байту за адресою location та повернення «0» у разі успіху.

Ініціалізація області EEPROM відбувається з використанням pragma-визначень.

Для ініціалізації EEPROM визначають символічні імена відповідних змінних та їх вміст. Нижче наведено приклад такої ініціалізації.

Приклад 5

```
#pragma data:eeprom
unsigned char ee_var1 = 10;
unsigned int  ee_var2 = 0x100;
#pragma data:data
```

Після компіляції проекту у цьому випадку генерується вихідний файл <output file>.eep.

Якщо розробляються власні функції запису/читання, то використовують наступну інформацію.

Для звертання до EEPROM - пам’яті використовуються три регістри: регістр адреси, регістр даних та регістр керування.

В регістр адреси завантажується адреса комірки, до якої будуть звертатися. Регістр адреси складається з двох байт – EEARH та EEARL.

The EEPROM Address Register – EEARH and EEARL

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	–	EEAR8	EEARH
	EEAR7	EEAR6	EEAR5	EEAR4	EEAR3	EEAR2	EEAR1	EEAR0	EEARL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	X	
	X	X	X	X	X	X	X	X	

Регістр даних EEPROM - пам’яті, називається EEDR. При запису в цей регістр завантажуються дані, які повинні розміщатися в EEPROM за адресою, що знаходиться в регістрах EEARH:EEARL. При зчитуванні в цей регістр поступають дані записані в EEPROM за адресою, що знаходиться в регістрах EEARH:EEARL.

The EEPROM Data Register – EEDR

Bit	7	6	5	4	3	2	1	0	
	MSB							LSB	EEDR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Регістр керування використовується для управління доступом до EEPROM - пам’яті. Цей регістр називається EECR.

The EEPROM Control Register – EECR

Bit	7	6	5	4	3	2	1	0	
	–	–	–	–	EERIE	EEMWE	EEWE	EERE	EECR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	X	0	

Окремі біти цього регістру мають наступне значення:

EERE - дозвіл читання з EEPROM;

EEWE - дозвіл запису до EEPROM;

EEMWE – керування дозволом запису до EEPROM;

EERIE - дозвіл переривання від EEPROM.

Нижче наведено приклади реалізації функцій запису/читання байту.

Приклад 6

```
;-----------------------------------------------------------
;Підпрограма запису в EEPROM
;-----------------------------------------------------------
void EEPROM_write(unsigned int uiAddress,unsigned char ucData)
{
while (EECR & (1<<EEWE));     //очікування кінця звертання
     {
     EEAR = uiAddress;        //встановлення адреси EEPROM
     EEDR = ucData;           //встановлення даних EEPROM
     EECR |= (1<<EEMWE);      //запуск майстер - стробу
     EECR |= (1<<EEWE);       //запуск запису
     };
}
```

Приклад7

```
;-----------------------------------------------------------
;Підпрограма читання з EEPROM
;-----------------------------------------------------------
unsigned char EEPROM_read (unsigned int uiAddress)
{
while (EECR & (1<<EEWE));     //очікування кінця звертання
     {
     EEAR = uiAddress;        //встановлення адреси EEPROM
     EECR |= (1<<EERE);       //дозвіл читання
     return EEDR;             //повернення даних
     };
}
```

7.5 **Приклад пробного проекту.**

У наведеному нижче прикладі проводиться така ініціалізація змінних:

- ee_var1 та ee_var2 визначаються у EEPROM - сегменті;
- c_var1 визначається у FLASH (ROM) - сегменті;
- tmp_SRAM визначається у SRAM-області.

З використанням бібліотечних функцій зчитуються значення змінних, що розташовані у EEPROM - сегменті.

Проводиться математична обробка змінних у відповідності з наступним виразом

result = c_var1*(tmp_ee1+tmp_ee2)-tmp_SRAM.

```
//-----------------------------------------------------------
//ICC-AVR application builder: 11.02.2019 12:15:00
// Target: M16
// Crystal: 10.000Mhz
//-----------------------------------------------------------
#include <iom16v.h>
#include <macros.h>
#include <eeprom.h>
//ініціалізація змінних у EEPROM
```

```
#pragma data:eeprom
unsigned char ee_var1 = 100;
unsigned int  ee_var2 = 0x100;
#pragma data:data
//ініціалізація змінних у FLASH (ROM)
const char c_var1 = 2;
//-------------------------------------------------------------
//Функції
//-------------------------------------------------------------
// Функція ініціалізації портів
void port_init(void)
{
 PORTA = 0xC0;          //ініціалізація портів
 DDRA   = 0xF0;
 PORTB = 0x00;
 DDRB   = 0x00;
 PORTC = 0x00;
 DDRC   = 0x00;
 PORTD = 0x00;
 DDRD   = 0x00;
}
//-------------------------------------------------------------
// Функція ініціалізації мікроконтролера
void init_devices(void)
{
CLI();                  //загальна заборона переривань
port_init();            //ініціалізація портів

MCUCR = 0x00;
GICR  = 0x00;
TIMSK = 0x00;           //timer interrupt sources
SEI();                  //загальний дозвіл переривань
}
// Головна функція
void main(void)
{
init_devices();         //Ініціалізація мікроконтролера
while(1)                //Головний цикл
{
unsigned int result, tmp_ee1, tmp_ee2;
unsigned int tmp_SRAM = 10;
tmp_ee1 = EEPROMread(ee_var1);//читання char-змінної ee_var1
EEPROM_READ((int)&ee_var2,tmp_ee2);//читання змінної ee_var2
result=c_var1*(tmp_ee1+tmp_ee2)-tmp_SRAM; //математична
//обробка
};
}
//-------------------------------------------------------------
```

КОМП'ЮТЕРНИЙ ПРАКТИКУМ №2.
ВИВІД І ВВІД СТАТИЧНИХ ТА ІМПУЛЬСНИХ СИГНАЛІВ У МІКРОКОНТРОЛЕРАХ AVR

1. Мета роботи:

- вивчення роботи портів, таймерів/лічильників та системи переривань мікроконтролера Atmega16;
- ознайомлення з методами виводу і вводу статичних та імпульсних цифрових сигналів;
- розробка тестових проектів;
- відпрацювання програм симулятивним методом у середовищі AVR Studio;
- програмування мікроконтролера лабораторного стенда.

2. Програма роботи:

2.1 Ознайомитися з роботою портів, таймерів/рахівників та системи переривань мікроконтролера Atmega16. Ознайомитися з регістрами процесора, які дозволяють працювати з цими ресурсами мікроконтролера;

2.2 Вивчити особливості виводу і вводу статичних та імпульсних цифрових сигналів;

2.3 З використанням мови Сі створити пробний проект з наступними фрагментами:

2.3.1 Виводу статичних цифрових сигналів;

2.3.2 Виводу імпульсних цифрових сигналів із заданим періодом повторення та тривалістю імпульсів;

2.3.3 Вводу статичних цифрових сигналів;

2.3.4 Вводу цифрових сигналів, що формуються контактною парою (з врахуванням тремтіння контактів);

2.4 Провести компіляцію та вилучити виявлені помилки;

2.5 Провести покрокове відпрацьовування програми з використанням програмного симулятора AVR Studio;

2.6 Запрограмувати мікроконтролер. ***Увага! Перше програмування провести в присутності викладача***;

2.7 Зробити висновки по роботі.

3. Завдання до лабораторної роботи

З використанням мови Сі розробити програми, у відповідності з п.2.3 лабораторної роботи. У табл. 2.1 розташовані вихідні дані, що відносяться до виводу (пп.. 2.3.1, 2.3.2) та вводу (пп..2.3.3, 2.3.4) цифрових сигналів.

Таблиця 2.1

№	П.2.3.1		П.2.3.2			П.2.3.3	П.2.3.4	
	Порт	Порт PC	Порт	T [c]	Ti [c]	Порт	Порт	Ttp [мс]
1	PA4	0b01011100	PA6	2,5	0,3	PB1	PB4	2
2	PA5	0xFF	PA7	1	0,5	PB3	PB1	3
3	PA6	0b11100011	PA4	5	4	PB4	PB3	1
4	PA7	0b01010101	PA5	3,5	2	PB1	PB4	4
5	PA4	0	PA6	4	0,5	PB3	PB1	5
6	PA5	0b11001100	PA7	1,5	0,9	PB4	PB3	6
7	PA6	0b00000001	PA4	1	0,1	PB1	PB4	8
8	PA7	8	PA5	3,2	2	PB3	PB1	7
9	PA4	0b10100000	PA6	2	1	PB4	PB3	9
10	PA5	10	PA7	1,3	1	PB1	PB4	4
11	PA6	0b01010000	PA4	1,9	1,2	PB3	PB3	5
12	PA7	0b00000001	PA5	1,2	1	PB4	PB1	2

Де:

- “Порт” - порт призначення;
- T - період повторення імпульсів;
- Ti - тривалість імпульсу;
- Ttp - час тремтіння контактів.

У стенді порти використовуються наступним чином.

Лінії порту A, що використовуються для виводу:

- PA6 – підключено світлодіод LED1;
- PA7 - підключено світлодіод LED2;
- PA5 - підключено світлодіод LIGHT;
- PA4 – підключено гудок BEEP.

Лінії порту B, що використовуються для вводу:

- PB1 – DOWN;
- PB3 – UP;
- PB4 – MODE.

4. Зміст звіту

4.1 Титульний листок із назвою роботи та переліком виконавців.

4.2 Тексти програм.

4.3 Висновки.

5. Контрольні запитання

5.1 Як проводиться ініціалізація ліній портів для роботи в якості приймачів та передавачів?

5.2 Визначте основні режими роботи таймерів-лічильників та особливості їх ініціалізації.

5.3 Які особливості системи переривань AVR-мікроконтролерів?

5.4 Визначте принципи побудови програм, у разі використання системи переривань.

5.5 Визначте основні типи цифрових сигналів.

5.6 Які особливості виводу статичних цифрових сигналів у мікроконтролера ATMega16? Дайте приклади виводу по одній лінії, та по всьому порту.

5.7 Як реалізується вивід періодичних імпульсних сигналів з використанням системи переривань?

5.8 Які особливості вводу статичних цифрових сигналів у ATMega16? Дайте приклади вводу по одній лінії, та по всьому порту.

5.9 Які особливості вводу цифрових сигналів, що формуються контактною парою?

6. Література

6.1 Евстифеев А.В. Микроконтроллеры AVR семейства Classic фирмы "Atmel". – М.: Издательский дом «Додэка - XXI», 2002.

6.2 Евстифеев А.В. Микроконтроллеры AVR семейства Tiny и Mega фирмы "Atmel". – М.: Издательский дом «Додэка - XXI», 2004.

7. Теоретичні відомості

7.1 Порти вводу/виводу

Мікроконтролер ATMega16 має 4 восьмибітові порти вводу/виводу (A, B, C, D). Кожна лінія портів може бути використана незалежно як для прийому, так і передачі цифрових сигналів.

Вхідні буфери портів побудовані за схемою тригера Шмітта. Для ліній, що мають конфігурацію вхідних, можливе підключення внутрішнього резистора опором 35…120 кОм між входом та шиною живлення V_{DD}.

Вихідні драйвери портів забезпечують однакові значення максимального струму навантаження для обох значень рівня вихідної напруги (I_{OL}=I_{OH}=40мА).

У мікроконтролері реалізовано алгоритм роботи з портами «читання/модифікація/запис», що дозволяє виконувати ряд операцій пов'язаних з виводом безпосередньо на лініях портів.

Більшість ліній портів, окрім безпосередніх функцій вводу\виводу цифрових сигналів, мають також альтернативні функції. Лінії портів використовуються в альтернативному режимі у випадках застосування відповідних вбудованих контролерів, наприклад таких як TWI, ADC.

7.1.1 **Регістри портів**

Звертання до портів відбувається через регістри вводу/виводу. В адресному просторі вводу/виводу для кожного порту відведено по 3 регістри:

- DDRx - напрямку роботи (передавач/приймач);

- PORTx – вихідних даних;
- PINx – вхідних даних.

Наприклад, для порту A - DDRA, PORTA, PINA.

Розряди цих регістрів мають назви:

- DDx7… DDx0 – для регістрів DDRx;
- Px7…Px0 – для регістрів PORTx;
- PINx7… PINx0 – для регістрів PINx.

Регістри PINx дозволяють здійснити доступ до фізичних значень сигналів на виводах порту. Відповідно вони доступні лише для читання. Регістри PORTx та DDRx доступні як для читання, так і для запису. Після рестарту мікроконтролера в регістри PORTx та DDRx записуються початкові нулеві значення. Це відповідає режиму приймача.

Запис у порт означає запис необхідного стану для кожного виводу порту у відповідний регістр даних порту PORTx. А читання стану порту виконується читанням або регістра даних порту PORTx, або регістра виводів порту PINx. При читанні регістра виводів порту PINx відбувається зчитування логічних рівнів сигналів, що присутні на виводах порту. А при читанні регістра даних порту відбувається зчитування даних, що знаходяться у регістрі PORTx.

Регістри портів розташовані в сегменті регістрів вводу/виводу (32 регістру на початку сегмента), для якого можливі бітові маніпуляції. Це суттєво спрощує роботу з окремими лініями портів. Можливе виконання наступних команд бітових маніпуляцій (наприклад, для порту A):

- Встановлення «1» значення на лінії порту PA1

```
PORTA |=(1 << PA1);
```

- Встановлення «0» значення на лінії порту PA1

```
PORTA &=~(1 < < PA1);
```

- Інверсія вихідного значення на лінії порту PA1

```
PORTA ^=(1 << PA1),
```

та розгалуження:

- Виконання операцій, якщо на лінії порту PA1 «0» значення

```
if (!(PINA & (1 << PA1))) {};
```

- Виконання операцій, якщо на лінії порту PA1 «1» значення

```
if (PINA & (1 << PA1)) {}.
```

7.1.2 **Конфігурування портів**

Під час конфігурування портів встановлюють такі властивості:

- задають напрямки передачі даних ліній портів (вхід або вихід);
- підключають/відключають внутрішні підтягуючі резистори.

Напрямок передачі даних визначається вмістом регістра передачі даних DDRx. Якщо розряд DDxn цього регістра встановлено в «1», відповідний n-

вивід порту є виходом. У протилежному випадку (встановлено «0»), відповідний n-вивід порту є входом.

Керування підтягуючим резистором здійснюється за допомогою регістра даних PORTx. Якщо розряд Pxn регістра PORTx встановлено в «1» і відповідний вивід порту є входом, між цим виводом та шиною живлення підключається підтягуючий резистор. Для того щоб відключити підтягуючий резистор, необхідно або скинути відповідний розряд регістра
PORTx, або зробити вивід порту виходом.

7.1.3 **Приклад ініціалізації порту**

Початкова ініціалізація портів є надзвичайно важливою процедурою. Ініціалізація повинна проводитися одразу після рестарту мікроконтролера. Затримка з її проведенням у деяких випадках може привести до виникнення аварійної ситуації у об'єкті керування. Визначення типів ліній портів та їх початкового стану проводиться на підставі аналізу схеми пристрою. Нижче приведено приклад ініціалізації порту А з конфігурацією ліній, що наведено у табл. 2.2. У відповідності до цієї таблиці визначаються константи ініціалізації ini_DDRA та ini_PORTA.

Таблиця 2.2

Лінії порту	**PA7**	**PA6**	**PA5**	**PA4**	**PA3**	**PA2**	**PA1**	**PA0**
Передавач	так	так	так	так	ні	ні	ні	ні
Вихідний рівень	1	1	0	0	-	-	-	-
Приймач	ні	ні	ні	ні	так	так	так	так
Підтяжка	ні	ні	ні	ні	ні	ні	так	так

У функції ініціалізації портів доцільно використовувати символічні імена констант. Це суттєво спрощує модифікацію програми при її адаптації до розробленої печатної плати. Для символічного опису використовують директиву *#define ім'я значення.* Під час опису констант ініціалізації портів доцільно вживати бінарний формат чисел що дає змогу більш оперативно оцінювати бітовий стан порту.

Приклад 1

```
//ICC-AVR application builder : 13.02.2019 20:03:15
//Target : M16
//Crystal: 10.000Mhz

#include <iom16v.h>
#include <macros.h>
//---------------------------
#define    ini_DDRA    0b11110000 //константи ініціалізації
#define    ini_PORTA   0b11000011 //згідно з даними табл.2.2
//---------------------------
//---------- Функція ініціалізації портів
void port_init(void)//ініціалізація порту A
{
```

```
DDRA  = ini_DDRA;
PORTA = ini_PORTA;
}
//----------------------------
//---------- Функція ініціалізації мікроконтролера
void init_devices(void)
{
CLI();                    //заборона переривань
port_init();              //ініціалізація портів
SEI();                    //дозвіл переривань
}
//----------------------------
//---------- Головна функція програми
void main(void)
{
init_devices();
while(1)
      {
      }
}
//----------------------------
```

7.2 Таймери-лічильники

Мікроконтролери, у залежності від моделі, мають у своєму складі від одного до трьох таймерів/лічильників загального призначення.

Таймер/лічильник T0 8-розрядний, у більшості моделей може використовуватися лише для відліку чи вимірювання часових інтервалів, або як лічильник зовнішніх подій. При переповненні лічильного регістру таймеру генерується запит на переривання. Два інших таймера (16-розрядний T1 та 8-розрядний T2) мають додаткові по відношенню до T0 функції. Обидва таймери можуть генерувати запит на переривання не тільки при переповненні лічильного регістру, а й при здійсненні ряду інших подій. Також таймери можуть працювати в якості широтно-імпульсних модуляторів. Таймер T2, крім того, може працювати в асинхронному (відносно тактового сигналу мікроконтролера) режимі. У складі усіх мікроконтролерів сімейства є також сторожовий таймер. Цей таймер використовується для захисту від зациклювання програми.

7.2.1 Режими роботи таймерів/лічильників

Кожний таймер/лічильник використовує одну або більше лінії вводу/виводу портів загального призначення мікроконтролера. Функції реалізовані цими виводами під час роботи з таймерами/лічильниками, являються альтернативними функціями портів.

Таймер/лічильник T0 може використовуватися для формування часових інтервалів або для підрахунку числа зовнішніх подій. До його складу входять блок керування а також 2 регістри - регістр керування TCCR0 та лічильний

регістр TCNT0. Прапор переповнення лічильного регістру таймеру TOV0 знаходиться в регістрі прапорів переривань від таймерів TIFR. Дозвіл або заборона переривань від таймера здійснюється установкою/скиданням прапору TOIE0 регістра TIMSK. У моделі мікроконтролера ATmega16 таймер/лічильник T0 також може працювати в режимі широтно-імпульсного модулятора (ШІМ).

Таймер/лічильник T2 може працювати в наступних режимах:

- режим таймеру. В цьому режимі він може використовуватися для формування часових інтервалів, а також виконувати визначені дії при рівності вмісту лічильного регістру з заданим значенням;
- режим ШІМ. У цьому режимі таймер/лічильник формує періодичну імпульсну послідовність із заданими частотою та тривалістю імпульсів.

До складу таймера/лічильника T2 входять два робочих регістра (лічильний регістр TCNT2 та регістр порівняння OCR2), 8-розрядний цифровий компаратор, регістр керування TCCR2, регістр стану для асинхронного режиму ASSR, а також блоки керування та синхронізації.

В даній роботі використовується таймер/лічильник T1, тому докладніше розглянемо саме його.

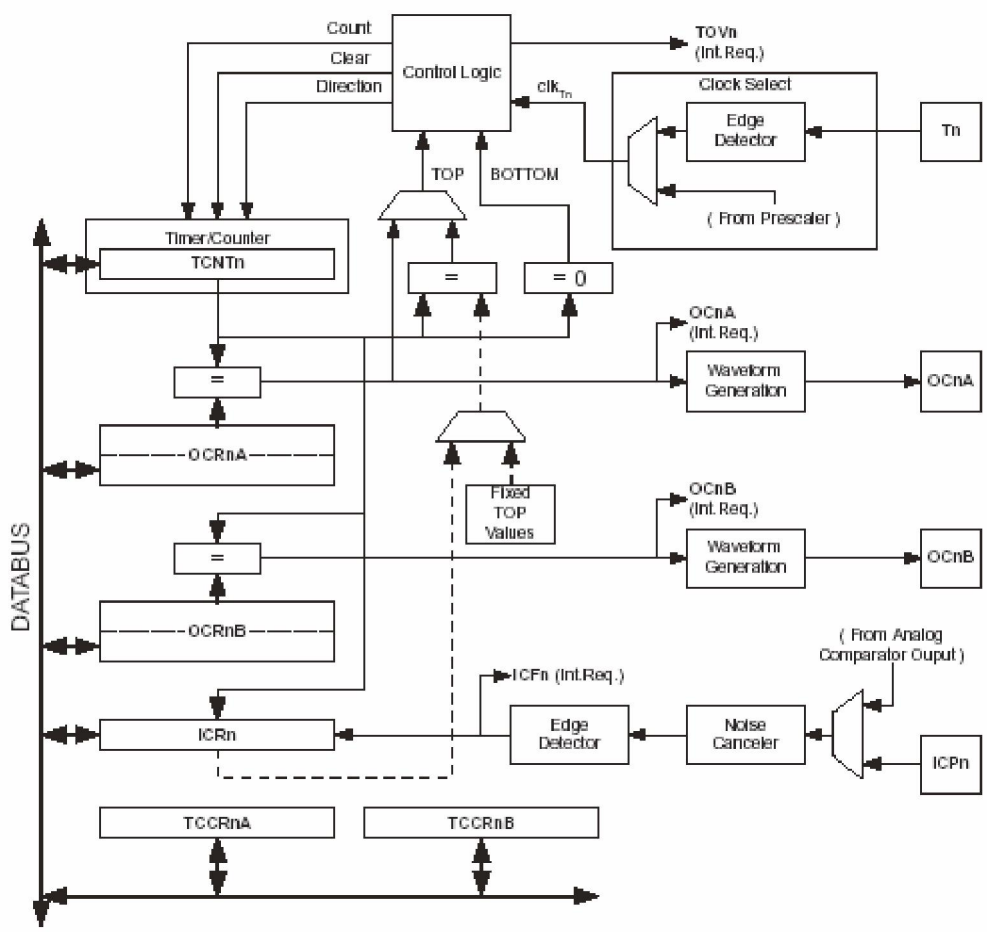
7.2.2 Таймер/лічильник T1

Таймер/лічильник T1 має наступні функції:

- може використовуватися для формування часових інтервалів або для підрахунку числа зовнішніх подій;
- може за зовнішнім сигналом зберігати свій стан в окремому РВВ;
- може виконувати визначені дії при рівності вмісту лічильного регістру з заданим значенням;
- може працювати як широтно-імпульсний модулятор.

До складу таймера/лічильника T1 входять чотири 16-розрядні регістри (лічильний регістр TCNT1H:TCNT1L, регістр захвату ICR1H:ICR1L та регістри порівняння OCR1AH:OCR1AL та OCR1BH:OCR1BL), 16-розрядний компаратор, два 8-розрядні регістри керування TCCR1A та TCCR1B, а також блок керування таймером.

Усі прапори стану таймера/лічильника (переповнення, збігу та захвату) знаходяться в регістрі прапорів переривань від таймерів TIFR, а дозвіл/заборона переривань від таймера здійснюється установкою/ скиданням відповідних прапорів регістра TIMSK. Нижче приведено структурну схему таймера/лічильника T1.



Лічильний регістр використовує операцію додавання (в режимі ШІМ – додавання/віднімання) та доступний в будь-який момент часу як для читання, так і для запису. Під час запису в регістр TCNT1, при роботі в режимі таймера, рахунок буде продовжений за наступним за операцією запису імпульсом тактового сигналу таймера/лічильника. Після подачі напруги живлення в регістрі TCNT1 буде знаходитися нульове значення.

Фізично регістр TCNT1 розміщений в двох регістрах TCNTH1:TCNTL1. Щоб при звертанні до цих регістрів запис або читання обох байтів відбувався одночасно, використовується спеціальний регістр TEMP (цей регістр використовується лише процесором та програмно недосяжний). Цей же регістр використовується і при звертанні до решти 16-розрядних регістрів таймера/лічильника T1: OCR1A, OCR1B та ICR1. Переривання на час звертання до цих регістрів має бути забороненим.

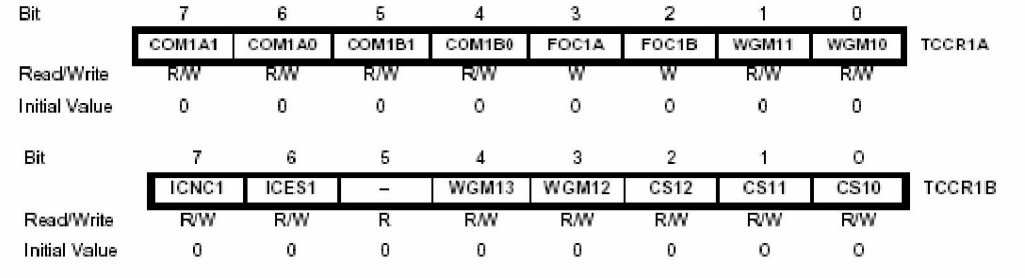
Запис у регістр TCNT1

При запису старшого байту значення в регістр TCNT1H він розміщується в регістрі TEMP. Далі, при запису молодшого байту в регістр TCNT1L він об’єднується з вмістом регістра TEMP та обидва байти записуються в регістр TCNT1 одночасно.

Читання регістра TCNT1

При читанні TCNT1L (молодший байт) вміст регістра TCNT1L пересилається в регістр TEMP. А при наступному читанні регістра TCNT1H повертається значення, що збереглося в регістрі TEMP.

Керування таймером/лічильником Т1 здійснюється за допомогою двох регістрів керування TCCR1A та TCCR1B. Нижче наведено бітову структуру цих регістрів.



7.2.3 Вибір джерела тактового сигналу

По відношенню до тактового сигналу таймер/лічильник може працювати в двох режимах:

- Режим таймера. В цьому режимі на вхід таймера/лічильника надходять імпульси тактового сигналу мікроконтролера (безпосередньо або через подільник);
- Режим рахування кількості подій. В цьому режимі інкремент вмісту лічильного регістру відбувається за активним фронтом сигналу на вході Т1 мікроконтролера (режим альтернативних функцій).

Вибір джерела тактового сигналу, а також запуск або зупинка таймера/лічильника здійснюється за допомогою розрядів CS12…CS10 регістра керування таймером TCCR1B.

Регістр TCCR1B			**Джерело тактового сигналу**
CS12	**CS11**	**CS10**	
0	0	0	Таймер/лічильник зупинено
0	0	1	СК – тактовий сигнал мікроконтролера
0	1	0	СК/8
0	1	1	СК/64
1	0	0	СК/256
1	0	1	СК/1024
1	1	0	Лінія порту Т1, інкремент по спадаючому фронту сигналу
1	1	1	Лінія порту Т1, інкремент по зростаючому фронту сигналу

При використанні зовнішнього тактового сигналу необхідно пам’ятати, що він синхронізується з частотою тактового генератора мікроконтролера. Тому для забезпечення коректної роботи таймера від зовнішнього сигналу проміжок часу між сусідніми імпульсами повинен бути більше періоду тактового сигналу мікроконтролера.

7.2.4 Режим таймера

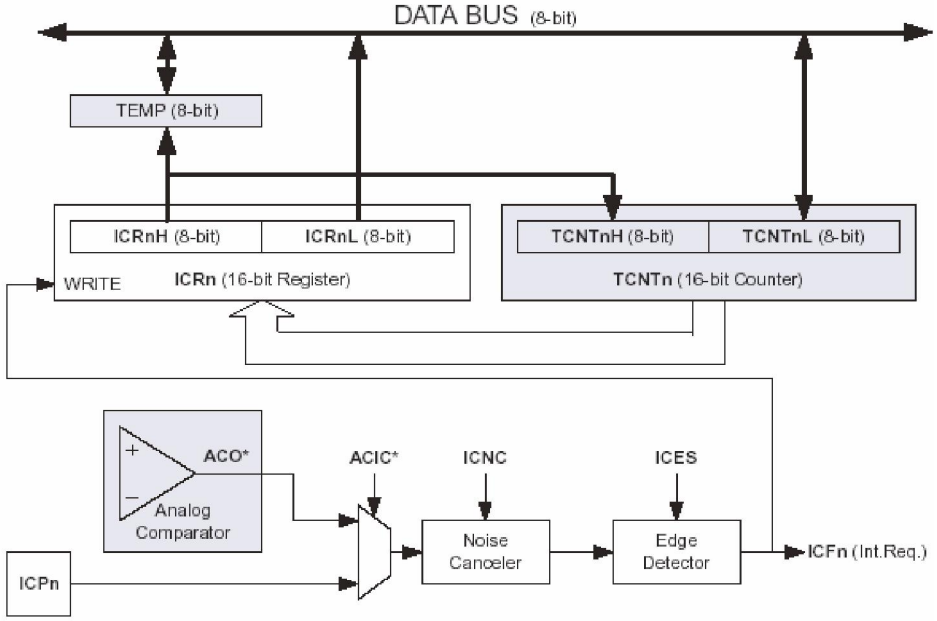
У цьому режимі роботи за кожним імпульсом, що надходить на тактовий вхід таймера/лічильника, відбувається інкремент вмісту лічильного регістру TCNT1. При переході таймера/лічильника зі стану «$FFFF» у стан «$0000» встановлюється прапор TOV1 регістра TIFR та генерується запит на переривання. Дозвіл переривання здійснюється установкою в «1» розряду TOIE1 регістра TIMSK (прапор I загального дозволу переривань регістра SREG також повинен бути встановлений в «1»).

Bit	7	6	5	4	3	2	1	0	
	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	TIMSK
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	TIFR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

7.2.4.1. Функція захвату (Capture)

Під даною функцією розуміють збереження у визначений час стану таймера/лічильника в регістрі захвату ICR1. Ця дія може виконуватися або за активним фронтом сигналу на виводі ICP мікроконтролера, або за сигналом від компаратору. При цьому встановлюється прапор ICF1 регістра TIFR та генерується запит на переривання. Дозвіл переривань
здійснюється встановленням в «1» розряду TICIE1 регістра TIMSK.

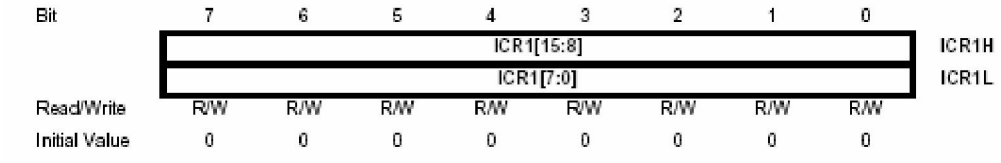


Для керування схемою захвату використовується два розряди регістра TCCR1B: ICNC1 та ICES1. Розряд ICNC1 керує схемою знешкодження завад. Якщо цей розряд встановлений в «0», схема знешкодження завад вимкнена та захват відбувається за першим активним фронтом на виводі ICP мікроконтролера. Якщо цей розряд встановлений в «1», то при надходженні

активного фронту на вивід ICP виконуються 4 вибірки з частотою, рівною тактовій частоті мікроконтролера. Захват виконується якщо всі вибірки мають рівень, що відповідає активному фронту сигналу («1» для зростаючого та «0» для спадаючого).

Активний фронт сигналу визначається станом розряду ICES1. Якщо цей розряд скинутий в «0», то активним є спадаючий фронт. Якщо ж цей розряд встановлений в «1», то активним є зростаючий фронт. Вихід ICP повинен мати конфігурацію вхідного виводу, тобто розряд керування портом DDRx, що відповідає даному виводу, має бути скинутим в «0».

Фізично регістр захвату ICR1 розміщений в двох регістрах ICR1H:ICR1L які доступні лише для читання.



Оскільки регістр захвату є 16-розрядним, при його читанні використовується спеціальний тимчасовий регістр TEMP. При читанні регістра ICR1L вміст цього регістра пересилається, а вміст регістра ICR1H зберігається в регістрі TEMP. При читанні регістра ICR1H повертається значення, що збережене в регістрі TEMP. Отже, при читанні регістра ICR1 першим повинен бути прочитаний регістр ICR1L. Переривання на час звертання до регістра ICR1 мають бути забороненими.

7.2.4.2. Функція порівняння (Compare)

Ця функція передбачає безперервне порівняння вмісту лічильного регістру таймера/лічильника з числом, що знаходиться в регістрі порівняння. При рівності вмісту в цих регістрах встановлюється прапор відповідного переривання, а також можуть виконуватися інші дії.

Якщо стан таймера/лічильника буде тотожнім стану регістру порівняння, то в наступному машинному циклі встановлюється відповідний прапор переривань в регістрі TIFR та генерується запит на переривання. Дозвіл переривань здійснюється встановленням в «1» відповідних прапорців регістра TIMSK.

Разом з установкою прапора в регістрі TIFR, при рівності вмісту лічильного регістра та регістра порівняння, можуть виконуватися й інші дії:

- скидання таймера/лічильника;
- зміна стану визначеного виводу мікроконтролера.

Поведінка мікроконтролера визначається відповідними бітами регістрів керування TCCR1A та TCCR1B.

Кожен регістр порівняння фізично розташований у двох регістрах вводу/виводу, які доступні для запису/читання:
OCR1A – OCR1AH:OCR1AL та OCR1B – OCR1BH:OCR1BL.

7.3 Система переривань

7.3.1 Загальні відомості

Мікроконтролери AVR мають систему переривань із одним фізичним рівнем пріоритету. Проте існує можливість логічної організації багаторівневої системи. Кількість джерел переривань залежить від типу мікроконтролера. Пріоритетність цих джерел визначається адресами їх векторів переривання. Чим менша адреса вектора, тим вищий пріоритет переривання. Це означає, що у разі одночасної генерації двох запитів першим буде оброблятися запит із меншою адресою вектора переривання.

Джерела переривань поділяються на два типи. Джерела першого типу формують запити переривань тригерного типу. Прапори переривань цих запитів зберігаються до моменту виконання підпрограми переривань, або їх програмного скидання. Джерела другого типу формують запит переривання у разі існування умов. Відсутня пам'ять таких запитів. Наприклад, якщо зовнішній сигнал що викликає запит переривання пропадає, то пропадає і запит.

Кожне переривання має власний біт керування дозволом. Крім того в регістрі SREG є біт загального дозволу переривань (I).

У разі виконання підпрограми переривань мікроконтролер записує в стек вміст програмного рахівника (PC) та замість нього розміщує адресу відповідного вектора переривань. Якщо переривання викликається запитом тригерного типу, то скидається прапор запиту цього переривання. За цією адресою у мікроконтролера ATMega16 має бути розташована команда переходу на початок підпрограми переривань (jmp addr). На час виконання цієї підпрограми апаратно скидається біт загального дозволу переривань, що забороняє обробку інших переривань. Завершується підпрограма переривань командою *RETI* за якою програмний рахівник завантажується зі стеку адресою точки виходу в переривання та поновлюється загальний дозвіл переривань.

7.3.2 Таблиця векторів переривань

Для мікроконтролера ATmega16 номери векторів переривань, символічні імена, та умови їх генерації приведено в табл. 2.3.

Звичайно ця таблиця розташовується за вказаними адресами. Проте існує можливість програмного переносу адрес векторів переривання на початок BOOT- сектору. У зв’язку з тим, що існує можливість зміни розмірів BOOT- сектору, відповідно можливі і різні варіанти розташування векторів переривання. У залежності від значень фюзесів BOOTSZ1:BOOTSZ0 можливі 4 варіанти розташування початку цієї секції.

Для програмної зміни адрес векторів переривання використовуються біти керування IVSEL та IVCE регістру керування GICR. Біт IVSEL визначає місце розташування таблиці в області прикладної програми

Таблиця 2.3

№	Адреса	Джерело	Визначення переривання
1	$000	RESET	Скидання мікроконтролера
2	$002	INT0	Зовнішнє переривання по входу INT0
3	$004	INT1	Зовнішнє переривання по входу INT1
4	$006	TIMER2COMP	Співпадання по таймеру/лічильнику T2
5	$008	TIMER2OVF	Переповнення таймера/лічильника T2
6	$00A	TIMER1CAPT	Захват таймера/лічильника T1
7	$00C	TIMER1COMPA	Співпадання по таймеру/лічильнику T1A
8	$00E	TIMER1COMPB	Співпадання по таймеру/лічильнику T1B
9	$010	TIMER1OVF	Переповнення таймера/лічильника T1
10	$012	TIMER0OVF	Переповнення таймера/лічильника T0
11	$014	SPI, STC	Завершення обміну по SPI
12	$016	USART, RXC	Прийом по USART закінчено
13	$018	USART, UDRE	Вивільнення регістру даних USART
14	$01A	USART, TXC	Передачу по USART закінчено
15	$01C	ADC	Завершення перетворення ADC
16	$01E	EE_RDY	Готовність EEPROM
17	$020	ANA_COMP	Аналоговий компаратор
18	$022	TWI	Переривання від контролера TWI
19	$024	INT2	Зовнішнє переривання по входу INT2
20	$026	TIMER0COMP	Співпадання по таймеру/лічильнику T0
21	$028	SPM_RDY	Завершення запису в пам'ять програм

(IVSEL =0) або на початку BOOT- сектора (IVSEL =1). Біт IVCE виконує функцію допоміжного майстер-біту. Для зміни положення таблиці необхідно спочатку встановити IVCE = 1, а потім протягом 4 циклів мікроконтролера змінити значення біту IVSEL. Після такої операції
майстер-біт автоматично скидається.

7.4 Вивід цифрових сигналів

7.4.1 Вивід статичних цифрових сигналів

Для виводу цифрових сигналів необхідно налаштувати відповідні лінії портів на режим передавача (п.7.1.3). Операції виводу проводять із регістрами PORTx (PORTA, PORTB, PORTC, PORTD).

Для встановлення лінії портів в “1”, або скидання в “0” - стан використовуються наступні команди:

- установка «1» PORTx |=(1 << bit);
- скидання в «0» PORTx &=~(1 < < bit);
- інверсія PORTx =(1 << bit),

де: “PORTx” – вихідний порт, “bit” – біт призначення.

Якщо весь порт використовується в режимі передавача, і одночасно змінюється велика кількість біт, то доцільно використовувати команди завантаження байтів

PORTx = const_1,

де: константа “const_1” попередньо визначається наступним чином

```
#define    const_1  0b11110000
```

Не допускається виконання логічних і арифметичних операцій безпосередньо на регістрах PORTx. Такі операції можливі лише з файловими регістрами.

7.4.2 Вивід імпульсних сигналів

Розрізняють наступні типи імпульсних сигналів:

- Одиночні сигнали;
- Одиночна послідовність ряду імпульсів (пакет імпульсів);
- Періодичні сигнали із установленими періодом і тривалістю імпульсів;
- Періодична послідовність пакетів імпульсів.

Для формування імпульсних сигналів можливе використання, як програмних засобів, так і можливостей таймерів-лічильників мікроконтролера.

7.4.2.1 Вивід імпульсних сигналів із використанням програмних засобів

Такий тип виводу використовується у випадках, коли відсутні істотні обмеження по часу виконання процедури, або при необхідності створення драйверів зовнішніх мікросхем із жорсткими вимогами до тривалості та періоду повторення імпульсів керування.

Загальний алгоритм виводу передбачає реалізацію виводу на порт певного логічного стану та необхідної програмної затримки до моменту виводу наступного стану.

У другому прикладі забезпечується формування на лінії порту LED1 періодичної послідовності двох негативних імпульсів з наступними параметрами:

- Тривалість першого та другого імпульсів – 1с;
- Затримка між першим і другим імпульсами – 500мс;
- Період повторення пачки імпульсів – 4с.

Приклад 2

```
//---------------------------
//ICC-AVR application builder : 20.01.2019 15:11:12
//Target : M16
//Crystal: 10.000Mhz

#include <iom16v.h>
#include <macros.h>
//---------- Символічні визначення ліній портів та макроси
#define    LED1    PA6    //лінія керування світлодіодом LED1

#define LED1_ON PORTA &=~(1<<LED1) //включення LED1
```

```
#define LED1_OFF PORTA |= (1<<LED1)//вимикання LED1
//-----------------------------
//---------- Функція ініціалізації портів
void port_init(void)
{
DDRA  = 0b11001111;      //визначення типу ліній портів
PORTA = 0b11110000;      //визначення початкового стану
}
//-----------------------------
//---------- Функція ініціалізації мікроконтролера
void init_devices(void)
{
CLI();                   //заборона переривань
port_init();             //ініціалізація портів
SEI();                   //дозвіл переривань
}
//-----------------------------
//--------- Функція програмної затримки на "t_mcs" мікросекунд
void DelayMcs (unsigned int t_mcs)
     {while(t_mcs--)
          {asm("nop"); asm("nop"); asm("nop"); asm("nop"); };}
//-----------------------------
//--------- Функція програмної затримки на "t_ms" мілісекунд
void DelayMs (unsigned int t_ms)
     {while(t_ms--) {DelayMcs(1000);};}
//-----------------------------
//--------- Функція програмної затримки на "t_s" секунд
void DelayS (unsigned int t_s)
     {while(t_s--) {DelayMs(1000);};}
//-----------------------------
//---------- Головна функція програми
void main(void)
{
init_devices();     //ініціалізація мікроконтролера
while(1)
     {
     LED1_ON;       //включення світлодіоду LED1
     DelayS(1);     //затримка 1 сек
     LED1_OFF;      //вимикання світлодіоду LED1
     DelayMs(500);  //Затримка 0.5 сек
     LED1_ON;       //включення світлодіоду LED1
     DelayS(1);     //Затримка 1 сек
     LED1_OFF;      //вимикання світлодіоду LED1
     DelayS(4);     //затримка 4 сек
     }
}
//-----------------------------
```

7.4.2.2 Вивід імпульсних сигналів з використанням таймерів-лічильників

У разі жорстких вимог до використання часу виконання основної програми мікроконтролера для формування імпульсних сигналів застосовують таймери-рахівники.

Звичайно за допомогою таймерів вирішують наступні типи задач:

- Формування тривалості затримки;
- Формування імпульсів з заданою тривалістю;
- Формування імпульсів з заданими тривалістю та періодом повторення.

Загальний алгоритм вирішення перших двох задач передбачає ініціалізацію обраного таймеру на реалізацію часу затримки, налаштування системи переривань та запуск таймеру.

Відмінності починаються у підпрограмі переривань. У першому випадку у підпрограмі переривань вимикається таймер. У другому випадку – виконуються також процедури маніпуляції з лініями портів, де формуються сигнали.

При вирішенні третього типу задач використовують PWM режим роботи. За рахунок ініціалізації таймера на певних портах мікроконтролера можливе безперервне формування PWM - сигналу.

Розглянемо два приклади реалізації задач другого та третього типів.

Приклад 3

У третьому прикладі забезпечується формування на лінії порту LED2 негативного імпульсу тривалістю 2с. Необхідна тривалість затримки забезпечується вибором коефіцієнту «попереднього подільника» 1/1024 таймера TC1 і завантаженням у робочі регістри TCNT1 відповідного початкового значення. Використано режим “Normal” таймера/рахівника.

```
//----------------------------
//ICC-AVR application builder : 11.02.2019 12:13:48
//Target : M16
//Crystal: 10.000Mhz

#include <iom16v.h>
#include <macros.h>
//---------- Символічні визначення ліній портів та константи
#define      LED2    PA7 //лінія керування світлодіодом LED2
#define      TC1_ON_1024 0b00000101     //включення TC1-1/1024
#define      TC1_OFF 0   //виключення TC1
//----------------------------
//---------- Вектор переривання переповнення TC1
#pragma interrupt_handler  INT_TC1_OVF: iv_TIMER1_OVF
//----------------------------
//---------- Функція ініціалізації портів
```

```
void port_init(void)
{
DDRA  = 0b11001111;      //визначення типу ліній портів
PORTA = 0b11110000;      //визначення початкового стану
}
//---------------------------
//---------- Функція ініціалізації TC1
void init_timer(void)
{
TIMSK  = 0x04; //дозвіл переривань переповнення TC1
TCNT1H = 0xb3; //завантаження TC1 на затримку 2 секунди
TCNT1L = 0xb4;
}
//---------------------------
//---------- Функція ініціалізації мікроконтролера
void init_devices(void)
{
CLI();                   //загальна заборона переривань
port_init();             //ініціалізація портів
init_timer();            //ініціалізація таймера TC1
SEI();                   //загальний дозвіл переривань
}
//---------------------------
//---------- Функція переривань переповнення TC1
void INT_TC1_OVF(void)
{
PORTA |= (1<<LED2);      //вимикання світлодіода LED2
TCCR1B = TC1_OFF;        //відключення таймера TC1
}
//---------------------------
//---------- Головна функція програми
void main(void)
{
init_devices();

PORTA &=~(1<<LED2);      //включення світлодіода LED2
TCCR1B = TC1_ON_1024;    //включення TC1 з подільником 1/1024
while(1){};
}
//---------------------------
```

Приклад 4

У четвертому прикладі забезпечується формування на лінії порту PWM (PD4 – OC1B) періодичної послідовності позитивних імпульсів з алгоритмом утворення “Fast PWM”.

Використана секція “B” таймера TC1, попередній подільник 1/256. Реалізовано режим “Fast PWM”, 8 розрядів, порт встановлюється в «1» на інтервалі формування імпульсу.

Форму сигналів можна перевірити за допомогою осцилографа (клема PWM стенду).

Параметри імпульсів:

- Частота імпульсної послідовності – 152,6 Гц. Частота визначається наступним чином

$$F_{OC}=f_{CLK} / (N*TOP),$$

де f_{CLK} – частота генератора мікроконтролера; N – коефіцієнт ділення попереднього подільника; TOP – модуль рахування. Для 8-бітової “Fast PWM” величина TOP складає 256;

- Тривалість – 3,25мкс;
- Розрядність (точність) PWM – 8 біт.

```
//----------------------------
//ICC-AVR application builder : 12.02.2019 12:33:45
// Target : M16
// Crystal: 10.000Mhz

#include <iom16v.h>
#include <macros.h>
//----------
#define   ini_TCNT1      0    //період T
//режим   PWM 8bit fast
#define   ini_TCCR1A     (1<<COM1B1)|(1<<COM1B0)|(1<<WGM10)
#define   ini_TCCR1B     (1<<CS12)|(1<<WGM12)
#define   ini_OCR1BL     (0x00ff-127) //тривалість імпульсу T/2
//----------------------------
//---------- Функція ініціалізації портів
void port_init(void)
{
DDRD  = 0b00010000; // лінія порту PWM (PD4 – OC1B) - передавач
PORTD = 0x00;
}
//----------------------------
//---------- Функція ініціалізації TC1
//TIMER1 initialize - prescale:256
//WGM: 5) PWM 8bit fast, TOP=0x00FF
//desired value: 153.8462Hz
//actual value: 152,588Hz (0,8%)
//Константи ініціалізації

void timer1_init(void)
{
TCCR1B = 0x00;          //зупинка таймера
TCNT1  = ini_TCNT1;     //завантаження періоду
OCR1BL = ini_OCR1BL;    //завантаження тривалості імпульсів
TCCR1A = ini_TCCR1A;    //завантаження режиму роботи
TCCR1B = ini_TCCR1B;    //старт таймера
```

```
}
//-----------------------------
//---------- Функція ініціалізації мікроконтролера
void init_devices(void)
{
CLI();                    //загальна заборона переривань
port_init();              //ініціалізація портів
timer1_init();            //ініціалізація таймера TC1
SEI();                    //загальний дозвіл переривань
}
//-----------------------------
//---------- Головна функція програми
void main(void)
{
init_devices();           //ініціалізація мікроконтролера
while(1){};
}
//-----------------------------
```

7.5 Ввід цифрових сигналів

Для вводу цифрових сигналів необхідно налаштувати відповідні лінії портів на режим приймача (п.7.1.3). Для вводу використовують регістри PINx (PINA - PIND).

Побудова алгоритмів вводу залежить від типу цифрових сигналів (статичні, імпульсні), а також від присутності флуктуацій форми сигналів. Розрізняють алгоритми вводу сигналів стабільної форми, наприклад конфігураційного поля джемперів, а також сигналів із флуктуаціями. Характерним прикладом в останньому випадку може бути ввід сигналу від контактної пари, з програмною фільтрацією тремтіння контактів, де логічний рівень у процесі перемикання може декілька разів змінюватися.

При вводі імпульсних сигналів звичайно вирішують наступні задачі:

- Реєстрації сигналів;
- Реєстрації кількості імпульсних сигналів;
- Вимірювання періоду або тривалості періодичних сигналів.

7.5.1 Ввід статичних цифрових сигналів

В AVR- мікроконтролерів відсутні команди бітового обміну між файловими регістрами й регістрами введення/виводу. Для читання портів використовують команди завантаження байтів

Rd = PINx;

де: “ Rd ” - один з файлових регістрів, або змінна.

Не допускається виконання логічних і арифметичних операцій безпосередньо на регістрах PINx. Такі операції можливі лише з файловими регістрами чи змінними.

Разом з тим існує можливість реалізації програмних розгалужень на підставі аналізу стану окремих ліній портів PINx, за рахунок використання команд if(PINx & (1<<bit)){}, switch(){}.

7.5.2 Ввід цифрових сигналів, що формуються контактною парою

Для вводу сигналів, що формуються контактною парою, часто використовують наступний алгоритм:

- Проводиться первинне опитування стану порту. У разі виявлення замкненого стану контактів формується затримка, час якої перевищує час тремтіння контактів;
- Проводиться вторинне опитування стану порту на підставі якого приймається рішення про стан контактної пари.

Нижче приведено приклад програми, де вводиться цифровий сигнал від кнопки S3 (порт PB4) з триваліcтю тремтіння контактів < 2 мс. Замкненому стану контактів відповідає “0” , а розімкненому “1” рівень на лінії порту MODE (PB4). У програмі не використовується система переривань.

У прикладі може бути застосована функція затримки «DelayMs (**unsigned int** t_ms)», що описана у другому прикладі. В якості реакції на натискання кнопки використане перемикання світлодіодів LED1, LED2.

Приклад 5

```
//----------------------------
//ICC-AVR application builder : 13.03.2019 12:22:25
//Target : M16
//Crystal: 10.000Mhz

#include <iom16v.h>
#include <macros.h>
//---------- Символічні визначення ліній портів
#define    LED1    6     //лінія керування світлодіодом LED1
#define    LED2    7     //лінія керування світлодіодом LED2
#define    S3      PB4   //лінія вводу стану кнопки S1
//----------------------------
//---------- Функція ініціалізації портів
void port_init(void)
{
DDRA  = 0b11001111;      //визначення типу ліній порту A
PORTA = 0b11110000;      //визначення початкового стану
DDRB  = 0b00000000;      //визначення типу ліній порту B
PORTB = 0b00000000;      //визначення початкового стану
}
//----------------------------
//---------- Функція ініціалізації мікроконтролера
void init_devices(void)
{
CLI();                   //загальна заборона переривань
port_init();             //ініціалізація портів
```

```
SEI();                    //загальний дозвіл переривань
}
//---------------------------
//--------- Функція програмної затримки на "t_mcs" мікросукунд
void DelayMcs (unsigned int t_mcs)
     {while(t_mcs--)
              {asm("nop"); asm("nop"); asm("nop"); asm("nop");  };}
//---------------------------
//---------- Головна функція програми
void main(void)
{
init_devices();              //ініціалізація мікроконтролера
while(1)
     {
     if (!(PINB & (1<<S3)))   // опитування стану кнопки S1
        {
        DelayMcs(4000);       //затримка 4мс
        if  (!(PINB & (1<<S3)))    //повторне опитування S1
            {
            PORTA |=(1<<LED1);     //вимикання LED1
            PORTA &=~(1<<LED2);    //включення LED2
            }
        else
            {
            PORTA |=(1<<LED2);     //вимикання LED2
            PORTA &=~(1<<LED1);    // включення LED1
            };
        };
     };
}
//---------------------------
```

7.5.3 Ввід імпульсних цифрових сигналів. Вимірювання тривалості імпульсів

Задача вимірювання тривалості імпульсних цифрових сигналів, або періоду їх повторення вирішується звичайно з використанням таймерів та системи переривань. Переривання використовують для реєстрації початку та закінчення імпульсу.

Використовується наступний алгоритм:

- Після реєстрації імпульсу вмикається таймер;
- Після завершення імпульсу таймер зупиняється.

Тривалість імпульсу визначається по відомій величині тривалості одного циклу, що реєструє таймер, та по зареєстрованій таймером кількості таких циклів.

Недоліком таймерів AVR- мікроконтролерів є відсутність ресурсів для зовнішнього керування операціями вмиканням/вимиканням. Можливі лише

програмні реалізації, або використання інших контролерів, наприклад, вбудованого компаратора. Це обмежує точність фіксації моментів початку та закінчення імпульсів. Для підвищення точності фіксації доцільно використовувати переривання по входу INT0, що мають найвищий внутрішній пріоритет, та, відповідно, швидкість реакції. Решту переривань мікроконтролера на час вимірювань необхідно заборонити.

Для фіксації початку імпульсу потрібно налаштувати переривання по спадаючому (наростаючому) фронту INT0 – входу. Після включення таймера необхідно змінити настройку типу фронту на протилежну – падаючий (наростаючий).

У шостому прикладі розглянуто програму вимірювання тривалості імпульсу. Сигнал подається на вхід INT0 (PD2). Початок імпульсу фіксується по спадаючому, а закінчення – по наростаючому фронту. Використовуються переривання по входу INT0. Перевіряється тривалість імпульсу. Якщо тривалість імпульсу перевищує 5120мкс –світлодіод вимикається. У цьому прикладі використано те, що у стенді на вхід INT0 поступають імпульси які формуються з напруги мережі живлення частотою 50Гц. За допомогою осцилографа ці імпульси можна дослідити на роз'ємі «IMP» стенду.

Приклад 6

```
//---------------------------
//ICC-AVR application builder : 14.03.2019 11:46:11
//Target : M16
//Crystal: 10.000Mhz

#include <iom16v.h>
#include <macros.h>
//---------- Символічні визначення констант та змінних
#define TC1_ON  0b00000101    //попередній подільник 1/1024
#define TC1_OFF 0             //виключення таймера
#define ini_INT0    (1<<INT0) //дозвіл переривань по INT0
#define ini_INT0_10 (1<<ISC01)//переривання по фронту "1-0"
#define ini_INT0_01 (1<<ISC00)|(1<<ISC01)  //по фронту "0-1"
#define LED2   PA7            //визначення лінії LED2

volatile unsigned int Result=0;//змінна результату вимірювань
//---------------------------
#pragma interrupt_handler int0_isr:2 //Вектор переривання INT0
//---------- Функція ініціалізації портів
void port_init(void)
{
DDRA  = 0b11000000; //визначення приймачів - передавачів
PORTA = 0b11000000; //початковий стан ліній порту
DDRD  = 0x00;       //увесь порт - приймачі
PORTD = 0x00;       //відсутні резистори підтяжки
}
```

```
//---------- Функція ініціалізації TC1
//Режим Normal, TOP=0xffff
void timer1_init(void)
{
TCCR1B = TC1_OFF;   //зупинка таймера
TCNT1  = 0;         //очищення рахункових регістрів
}
//---------- Функція переривань INT0
void int0_isr(void)
{
if  (MCUCR == ini_INT0_10)    //розгалуження початок-кінець
    {
    TCCR1B = TC1_ON;          //включення таймера TC1
    MCUCR = ini_INT0_01;      //переривання по фронту 0-1
    }
else
    {
    TCCR1B = TC1_OFF;         //відключення таймера TC1
 Result = TCNT1;              //читання результату вимірювань
 MCUCR = ini_INT0_10;         //переривання по фронту 1-0
    TCNT1 = 0;                //очищення рахункових регістрів
    }
}
//---------- Функція ініціалізації мікроконтролера
void init_devices(void)
{
CLI();                   //загальна заборона переривань
port_init();             //ініціалізація портів
timer1_init();           //ініціалізація таймера

GICR  = ini_INT0;        //дозвіл переривань по входу INT0
MCUCR = ini_INT0_10;     //переривання по фронту "1-0"INT0
SEI();                   //загальний дозвіл переривань
}
//---------- Головна функція програми
void main(void)
{
init_devices();              //ініціалізація мікроконтролера
while(1)
  {
  if  (Result > 50) //перевірка результатів вимірювань
      {PORTA |=(1<<LED2);}   //виключити LED2, якщо тривалість
//більше 50*1024*0,1=5120мкс
  else
      {PORTA &=~(1<<LED2);}; //включити LED2, якщо менше
  };
}
//---------------------------
```

КОМП'ЮТЕРНИЙ ПРАКТИКУМ №3.
ВИВІД ІНФОРМАЦІЇ НА LED-ІНДИКАТОРИ

1. Мета роботи:

- ознайомлення з методами виводу інформації на LED-індикатори;
- засвоєння методу статичної індикації;
- засвоєння методу динамічної індикації;
- розробка тестових проектів;
- програмування мікроконтролера лабораторного стенду.

2. Програма роботи:

2.1 Ознайомитися з різновидами LED-індикаторів, особливостями побудови електричних схем та алгоритмів виводу інформації.

2.2 Для метода статичної індикації розробити програму, що забезпечує вивід символів у задані розряди багаторозрядного семисегментного індикатора, у залежності від натискання певних кнопок.

2.3 Для метода динамічної індикації розробити програму, що забезпечує вивід заданого числа у багаторозрядний семисегментний індикатор, у залежності від натискання певних кнопок. Розробити програми динамічної індикації з використанням програмних та таймерних методів формування затримки (використати переривання по таймеру).

2.4 Завантажити мікроконтролер програмами.

Увага! Перше програмування провести у присутності викладача.

2.5 Зробити висновки по роботі.

3. Завдання до лабораторної роботи

З використанням мови Сі необхідно розробити програми мікроконтролера, у відповідності з пп. 2.2, 2.3 та відпрацювати їх за допомогою лабораторного стенду.

В табл. 3.1 розташовані варіанти вихідних даних.

Таблиця 3.1

№	П.2.2						П.2.3			
	Кнопка	Розряд	Цифра	Кнопка	Розряд	Цифра	Кнопка	Число	Кнопка	Число
1	S1	ONE	1	S2	TEN	9	S3	2.30	S1	124
2	S2	TEN	2	S3	HUND	8	S1	12.3	S2	57
3	S3	HUND	3	S1	ONE	7	S2	0.89	S3	178
4	S1	ONE	4	S2	HUND	6	S3	0.23	S1	103
5	S2	ONE	5	S3	TEN	4	S1	2.01	S2	253
6	S3	TEN	6	S1	HUND	5	S2	19.7	S3	214
7	S1	HUND	7	S2	ONE	3	S3	2.35	S1	170
8	S2	TEN	8	S3	HUND	2	S1	17.4	S2	45
9	S3	ONE	9	S1	TEN	1	S2	0.73	S3	178
10	S1	TEN	1	S2	HUND	8	S3	22.9	S1	123
11	S2	HUND	2	S3	ONE	7	S1	1.42	S2	253

У стенді використовуються наступні елементи керування LED-дисплеєм:

- Лінія керування розрядом сотень HUND (PD5) - L-активна;
- Лінія керування розрядом десятків TEN (PD6) - L-активна;
- Лінія керування розрядом одиниць ONE (PD7) - L-активна;
- Шина даних (PORTC);
- Лінія стробування регістру шини даних LED_CS (PD3) – запис інформації у регістр відбувається при подачі позитивного імпульсу;
- Використовується пряма упаковка сегментів індикатора - A (PC0), B (PC1),…, DP (PC7);
- Кнопки S1 (PB1), S2 (PB3), S3 (PB4) - L-активні.

Увага! Для блокування LCD-індикатора необхідно подати на лінію LCD_CS(PA3) логічну «1».

4. Зміст звіту

4.1 Титульний листок із назвою роботи та переліком виконавців.

4.2 Тексти програм.

4.3 Висновки.

5. Контрольні запитання

5.1 Визначити типи індикаторів.

5.2 Визначити типи LED-індикаторів.

5.3 У чому полягає метод статичної індикації, та як він реалізується схемотехнічно?

5.4 У чому полягає метод динамічної індикації, та як він реалізується схемотехнічно?

5.5 Яким чином готуються дані для виводу на багаторозрядні семисегментні індикатори?

5.6 Які особливості програм драйверів багаторозрядних семисегментних індикаторів для метода статичної індикації?

5.7 Які особливості програм драйверів багаторозрядних семисегментних індикаторів для метода динамічної індикації?

6. Література

6.1 Евстифеев А.В. Микроконтроллеры AVR семейства Classic фирмы “Atmel”. – М.: Издательский дом «Додэка - XXI», 2002.

Евстифеев А.В. Микроконтроллеры AVR семейства Tiny и Mega фирмы “Atmel”. – М.: Издательский дом «Додэка - XXI», 2004.

7. Теоретичні відомості

7.1 **Класифікація індикаторів**

У залежності від типу фізичних процесів, що використовуються, розрізняють наступні види індикаторів - лампи накалювання; вакуумні,

люмінісцентні, світлодіодні (LED), рідкокристалічні (LCD), плазмові панелі, активні та пасивні матриці.

У мікроконтролерних системах керування та обробки інформації найбільш розповсюдженими є LED та LCD індикатори.

7.1.1. Індикатори LED - типу

Такі індикатори використовують при побудові дисплеїв, що працюють у широкому діапазоні температур (-50 - +100°С).

Індикатори характеризуються високою надійністю, яскравістю (у окремих зразків фірми Luxeon Star до 200 Кд), малою вартістю. Фірми Kingbright та Paralight виробляють індикатори з сімома основними кольорами випромінювання.

У LED-індикаторів звичайно є лише елементи індикації, а інтерфейсні вузли відсутні.

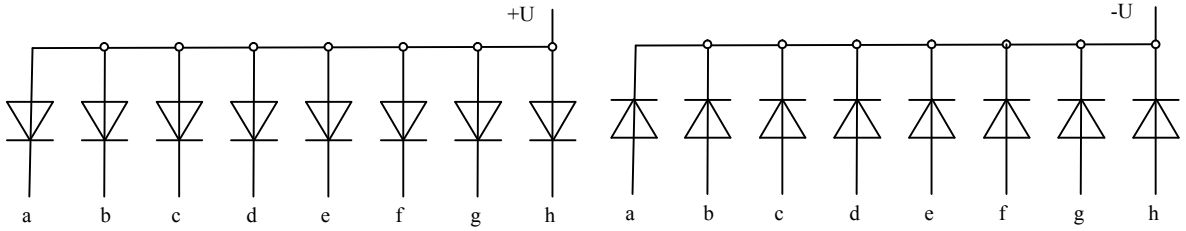
Виробляються індикатори із такими конструкціями:

- одиночні індикатори циліндричної форми із діаметрами 1, 3, 5, 7, 10мм та кластери;
- мнемонічні індикатори (прямокутної, трикутної, символьної форми);
- світлові табло прямокутної та круглої форми;
- графічні лінійки;
- одиночні семисегментні індикатори та їх набори (2-10 десяткових розрядів);
- мозаїчні індикатори.

Випромінювання LED-індикатора викликається протіканням прямого струму. Величина падіння напруги на прямо зміщеному світлодіоді залежить від величини струму та використаної технології, що визначає колір випромінювання.

Одиночні індикатори звичайно мають один світлодіод. В індикаторах з великою площею поверхні (табло, семисегментні індикатори великого розміру) можуть використовуватися декілька послідовно ввімкнених світлодіодів.

У дисплеях для відтворення цифр використовують семисегментні індикатори. У залежності від електричної схеми розрізняють два типи таких індикаторів – із спільним анодом (рис.3.1,а) та із спільним катодом (рис. 3.1,б).



а) б)

Рис.3.1

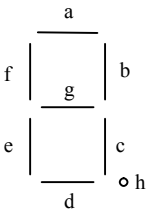


Рис. 3.2

На вибір типу індикатора впливають особливості принципової схеми пристрою індикації та допустимі рівні вихідного струму (високого та низького рівня) буферних підсилювачів. Сегменти індикатору позначаються літерами “a – h”. Звичайно використовується варіант розміщення сегментів, що приведений на рис. 3.2.

Для побудови багаторозрядних дисплеїв можуть застосовуватися одиночні семисегментні індикатори, або їх набори. У залежності від електричної схеми розрізняють набори призначені для статичної та динамічної індикації.

Набори семисегментних індикаторів для статичної індикації мають для кожного розряду числа повний комплект ліній керування (рис. 3.1).

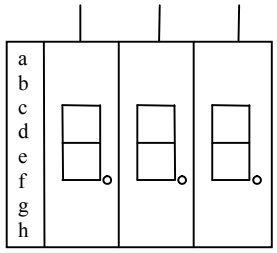


Рис.3.3

У наборах, що використовуються у схемах динамічної індикації (рис.3.3), лінії керування сегментами “a - h” усіх розрядів об`єднані. Розділяються лише лінії живлення розрядних груп (анодних або катодних).

Мозаїчні індикатори використовують у разі необхідності відображення графічних об’єктів, наприклад символів алфавіту. Такі індикатори побудовані на базі матриці індикаторів. Мозаїчні індикатори використовується для побудови дисплеїв із великою кількістю пікселей. Розмір таких індикаторів 4x4, 8x8, 10x10 світлодіодів. Електричну схему мозаїчного індикатора розміром 2x2 приведено на рис.3.4.

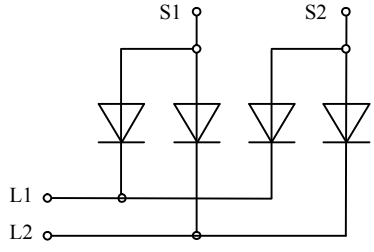


Рис. 3.4

Мозаїчні індикатори можуть використовуватися лише у дисплеях із динамічною індикацією.

7.2. Метод статичної індикації

У дисплеях, що використовують метод статичної індикації, через світлодіоди індикатора протікає безперервний робочий струм. Для забезпечення такого метода кожен одиночний сегмент індикатора необхідно підключати через власний буферний підсилювач. В якості таких підсилювачів досить часто використовують паралельні регістри. Для передачі інформації на

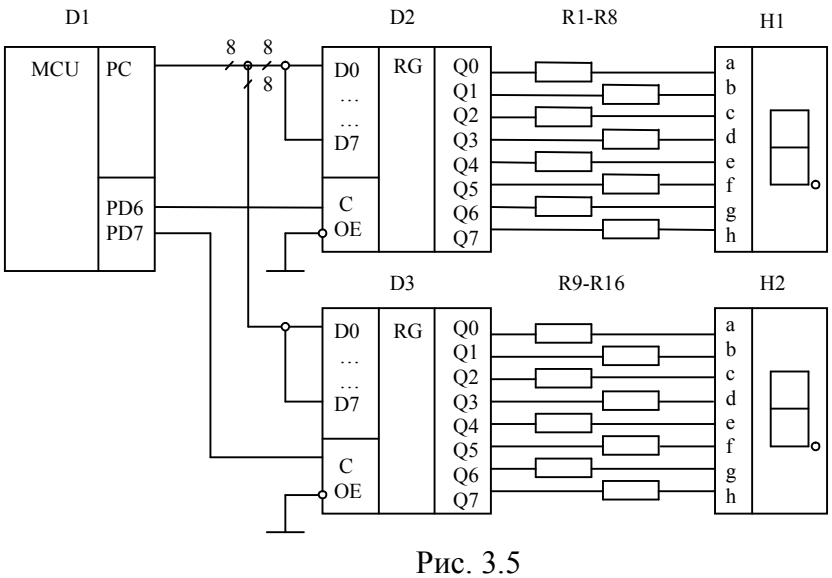


Рис. 3.5

регістри використовується шина P8 мікроконтролера (є у декілька AVR-процесорів з апаратним контролером P8), або ця шина формується програмний шляхом.

На рис.3.5 приведено принципову схему дворозрядного індикатора із програмним формуванням шини зв'язку з буферними регістрами. У схемі використано два буферних регістра D2, D3, підключених до загальної паралельної шини, яка формується з використанням порту PC. Для активізації виходів регістрів на лінії керування OE заведені активні логічні рівні -«0». Запис інформації у регістри одиниць та десятків відбувається за рахунок формування відповідних сигналів керування на лініях портів PD6, PD7. Резистори R1 – R16 використовуються для обмеження струму, що протікає через світлодіоди індикаторів H1, H2.

Використання метода статичної індикації спрощує програму-драйвер індикатора, але при цьому за рахунок великої кількості лінії керування та додаткових регістрів суттєво ускладнюється печатна плата.

7.3. Метод динамічної індикації

У дисплеях, що використовують динамічний метод індикації, через світлодіоди протікає імпульсний струм. Використовується спільний буферний регістр-підсилювач для керування сегментами "a - h" індикатора. Напруга живлення по черзі поступає на розряди індикатора. Синхронно із напругою живлення розрядів визначається і стан сегментів цих розрядів. Таким чином інформація по черзі виводиться у розряди індикатора. Для вилучення оптичних ефектів частота комутації кожного розряду повинна перевищувати 25 Гц. Світлодіоди індикаторів доцільно використовувати на частотах комутації до 0,5-4 кГц. Для забезпечення нормальної яскравості випромінювання сегментів необхідно підвищувати імпульсний струм світлодіодів так, щоб середнє значення струму відповідало номінальному статичному струму. Так у трирозрядному індикаторі необхідно підвищувати імпульсний струм у три рази, у порівнянні з відповідним індикатором статичного типу.

Динамічний метод індикації реалізується схемотехнічно простіше, у порівнянні із статичним, але програмний продукт більш складний.

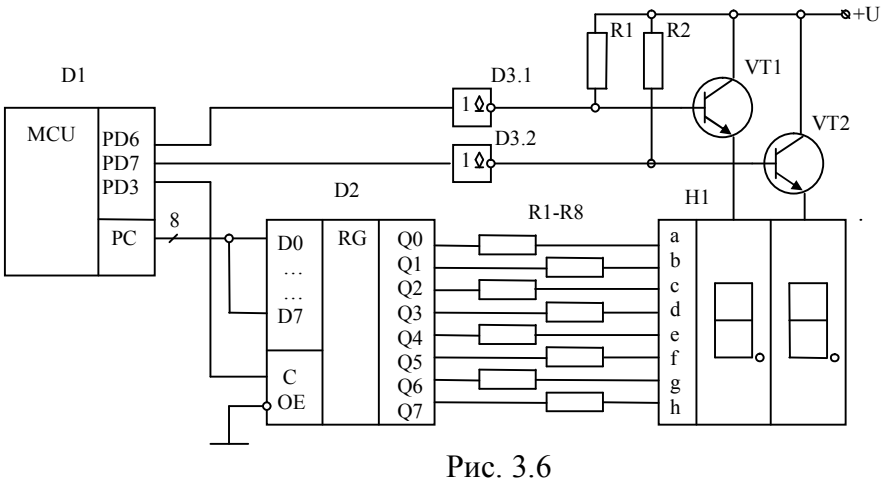


Рис. 3.6

На рис.3.6 приведено принципову схему дворозрядного індикатора, у якому використано метод динамічної індикації. У схемі використано

один буферний регістр-підсилювач, який підключено до паралельної шини мікроконтролера (порт PC). Для активізації виходів регістра на лінію OE подано низький логічний рівень напруги. Активізація відповідних розрядів індикатора реалізується за допомогою ліній процесора PD6, PD7. Для запису інформації з паралельної шини у регістр використовується сигнал керування, що формується на лінії PD3 мікроконтролера. Резистори R1 – R8 використовуються для обмеження струму світлодіодів індикатора. Для нормалізації напруги, що прикладається до світлодіодів індикатора використано підсилювачі напруги D3.1, D3.2. Транзистори VT1, VT2 використовуються для комутації розрядів індикатора.

7.4. Підготовка даних для виводу на семисегментний індикатор

Для виводу цифр на семисегментний індикатор необхідно провести їх перекодування. Визначення кодів залежить від упаковки сегментів в індикаторі, упаковки паралельної шини даних та від того, який вихідний рівень порту паралельної шини є активним для індикатора. У табл.3.2 приведено приклад визначення кодів «0» та «2» для схеми розташування сегментів, що наведено на рис. 3.2 та визначеного у таблиці порядку пакування бітів порту у індикатора з спільним анодом (LOW-активних рівній керування сегментами).

Таблиця 3.2

Сегменти індикатора	h	g	f	e	d	c	b	a
Упаковка ліній шини	BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
Код «0»	1	1	0	0	0	0	0	0
Код «2»	1	0	1	0	0	1	0	0

Коди цифр визначають у масиві та розміщують в пам’яті програм.

Приклад 1

```
//----------------------------
const unsigned char digits[] = {0b11000000, 0xF9, 0b10100100,
0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80,  0x90};
//----------------------------
```

У другому прикладі приведено програму відтворення цифри «2» у розряді десятків трирозрядного LED-індикатора. Код цифри «2» передається на вихід шини даних. Для перезапису цього коду на вихід буферного регістру формується строб керування регістром «CS_LED». Після цього активізується лінія вибірки розряду десятків індикатора. Цей приклад відповідає схемотехніці стенду до лабораторних робіт.

Приклад 2

```
PORTC = digits[2];  //передача на шину даних коду цифри «1»
PORTD |= (1<<CS_LED);    //формування стробу запису коду у
PORTD &= ~(1<<CS_LED);   //буферний регістр
PORTD &=~(1<<TEN);       //активізація розряду сотень
```

Звичайно необхідно передавати на дисплей багаторозрядні десяткові числа. У таких випадках використовують двійково-десяткове (BCD) перетворення з подальшим перекодуванням символів розрядів.

Нижче приведено функцію BCD – перетворення двобайтних чисел. Передача багаторозрядного десяткового числа у функцію реалізується за допомогою змінної «out», результати перетворення (цифри, що відповідають десятковим значенням числа) розташовані у змінних «ones», «tens»,«hunds».

Визначення десяткових розрядів реалізується за рахунок виділення у числі значень сотень десятків та одиниць.

Приклад 3

```
unsigned char ones;   //значення одиниць
unsigned char tens;   //значення десятків
unsigned char hunds;  //значення сотень
void transform(int out)
{
hunds =  out / 100;
tens  = (out % 100) / 10;
ones  =  out % 10;
}
```

7.4.1. Драйвер індикатора з статичною індикацією

У четвертому прикладі приведено програму керування трирозрядним індикатором лабораторного стенду із використанням методу статичної індикації.

У схемі використано буферний регістр D6 із статичним керуванням. Запис інформації відбувається при подачі на вхід “C” сигналу LED_CS (порт PD3) з високим рівнем напруги. Регістр D2 з`єднано з шиною катодів індикатора. Коди цифр визначались для семисегментного індикатора з загальним анодом (L-активні рівні напруги), для схеми розташування сегментів, що приведено на рис. 3.2. У першому прикладі приведено масив кодів цифр від «0» до «9».

Для активації розрядів сотень, десятків одиниць індикатора використовуються лінії керування “HUND”, “TEN”, “ONE” (активним є L-рівень напруги).

У прикладі на індикатор у розряд десятків виводиться число «6».

Приклад 4

```
//----------------------------
//ICC-AVR application builder : 14.02.2019 11:14:28
//Target : M16
//Crystal: 10.000Mhz

#include <iom16v.h>
#include <macros.h>
//---------- Символічні визначення ліній портів та константи
#define   TEN       PD6  //розряд десятків
```

```
#define   LED_CS    PD3  //строб запису в буферний регістр
//---------- Масив кодів цифр
const unsigned char digits[] = {0b11000000, 0xF9, 0b10100100,
0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80,  0x90};
//---------- Функція ініціалізації портів
void port_init(void)
{
PORTC = 0xFF;            //початковий стан всі "1"
DDRC  = 0xFF;            //налаштування порту C на передачу
PORTD = 0b11100000;      //початковий стан ліній порту
DDRD  = 0xFF;            //налаштування порту D на передачу
}
//---------- Функція ініціалізації мікроконтролера
void init_devices(void)
{
CLI();                   //загальна заборона переривань
port_init();             //ініціалізація портів
SEI();                   //загальний дозвіл переривань
}
//---------- Головна функція програми
void main(void)
{
init_devices();          //ініціалізація мікроконтролера
//---------- Вивід числа «6» у розряд десятків індикатора
PORTC = digits[6];       //встановити код символу
PORTD |= (1<<LED_CS);    //строб запису у регістр
PORTD &= ~(1<<LED_CS);
PORTD &=~(1<<TEN);       //активація розряду десятків
//--------
while(1)
     {};                 //нескінченний цикл
}
//---------------------------
```

Драйвер трирозрядного індикатора з динамічною індикацією

У п’ятому прикладі приведено програму керування трирозрядним індикатором лабораторного стенду із використанням методу динамічної індикації.

У програмі забезпечено вивід на індикатор довільного числа, значення якого не перевищує «999». Для виводу число передається як змінна функції BCD - перетворення `void transform(int out)`.

Для забезпечення стабільної частоти індикації розрядів використано таймер рахівник та систему переривань.

Індикація розрядів забезпечується програмними фрагментами, що розташовані у функції переривань.

Виділення значень десяткових розрядів забезпечується підпрограмою BCD-перетворення, яка описана у четвертому прикладі.

Приклад 5

```
//Crystal: 10.000Mhz
#include <iom16v.h>
#include <macros.h>
//---------- Символічні визначення ліній портів та константи
#define   ONE       PD7  //розряд одиниць
#define   TEN       PD6  //розряд десятків
#define   HUND      PD5  //розряд сотень
#define   LED_CS    PD3  //строб запису в буферний регістр
//---------- Масив кодів цифр
const unsigned char digits[] = {0b11000000, 0xF9, 0b10100100,
0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80,  0x90};
//---------- Визначення змінних
unsigned char indicator  = 1;      //змінна поточних розрядів
//1 - індикація одиниць; 2 – десятків; 3 - індикація сотень
unsigned char ones;                //значення одиниць
unsigned char tens;                //значення десятків
unsigned char hunds;               //значення сотень
//---------- Функція ініціалізації портів
void port_init(void)
{
PORTA = 0;              // початковий стан
DDRA  = 0;              // налаштування порту A
PORTC = 0xFF;           // початковий стан всі "1"
DDRC  = 0xFF;           // налаштування порту C на передачу
PORTD = 0b11100000;     // початковий стан ліній порту
DDRD  = 0xFF;           // налаштування порту D на передачу
}
//---------- Функція ініціалізації таймера
void timer0_init(void)
{
TCCR0 = 0x00;           //зупинка
TCNT0 = 0xF9;           //константа відліку
OCR0  = 0xFA;           //спрацювання таймеру
TCCR0 = 0x05;           //старт таймеру
}
//---------- Функція BCD - перетворення
void transform(int out)
{
hunds = out / 100;
tens = (out % 100) / 10;
ones = out % 10;
}
//---------- Функція переривань переповнення таймера T0
#pragma interrupt_handler timer0_ovf_isr:10
void timer0_ovf_isr(void)
{
indicator++;        //модифікація поточного розряду індикації
```

```
switch (indicator)
        {
        case 1:                         //індикація одиниць
               {
               PORTC = digits[ones]; //встановити код символу
               PORTD |= (1<<LED_CS); //формування строб
               PORTD &= ~(1<<LED_CS);
               PORTD |= (1<<HUND);   //активізація розряду
               PORTD &=~(1<<ONE);
               }; break;
       case 2:                          //індикація десятків
              {
               PORTC = digits[tens];
               PORTD |= (1<<LED_CS);
               PORTD &= ~(1<<LED_CS);
               PORTD |= (1<<ONE);
               PORTD &=~(1<<TEN);
               }; break;
       case 3:                          //індикація сотень
              {
               PORTC = digits[hunds];
               PORTD |= (1<<LED_CS);
               PORTD &= ~(1<<LED_CS);
               PORTD |= (1<<TEN);
               PORTD &=~(1<<HUND);
               indicator = 0;
               }; break;
       };
TCNT0 = 0xF9;                        //завантаження таймеру
}
//---------- Функція ініціалізації мікроконтролера
void init_devices(void)
{
CLI();
port_init();   //ініціалізація портів
timer0_init(); //ініціалізація таймеру
TIMSK = 0x01;  //дозвіл переривань переповнення таймера T0
SEI();         //загальний дозвіл переривань
}
//---------- Головна функція програми
void main(void)
{
init_devices();     //ініціалізація мікроконтролера
transform (145);    //BCD- перетворення числа «145»
while(1)            // нескінченний цикл
     {};
}
//---------------------------
```

КОМП'ЮТЕРНИЙ ПРАКТИКУМ №4.
ВИВІД ІНФОРМАЦІЇ НА СИМВОЛЬНІ LCD-ІНДИКАТОРИ

1. Мета роботи:

- ознайомлення з символьними LCD – індикаторами;
- ознайомлення з схемами підключення та методами керування;
- розробка проекту.

2. Програма роботи:

2.1 Засвоїти процедуру ініціалізації індикатора.

2.2 Скласти програму виводу текстового повідомлення з використанням латинського шрифту. Вивести текст у заданий рядок.

2.3 Скласти програму виводу текстового повідомлення з використанням літер російської/української мови. Використати утиліту HD44780.exe. Вивести текст у заданий рядок.

2.4 Скласти програму виводу комбінованого повідомлення (тексту та числа). Наприклад, для шостого варіанту повідомлення «Fosc=100Гц» виводиться у другий рядок, починаючи з третього знакомісця.

2.5 Перевірити виконання програм на лабораторному стенді.

Увага! Перше програмування провести у присутності викладача;

2.6 Зробити висновки по роботі.

3. Завдання до лабораторної роботи

У таблиці 4.1 розташовані варіанти вихідних даних.

У стенді використовуються наступні елементи керування:

- Лінія керування "LCD_CS" індикатора (PA3);
- Лінія керування "RS" індикатора (PD7);
- Лінія керування "RW" індикатора (PD6);
- Шина даних – порт PC;
- Лінія керування світлодіодом LIGHT (PA5, H - активна).

Таблиця 4.1

№	П.2.2		П.2.3		П.2.4				
	Ряд	Текст	Ряд	Текст	Ряд	Поз.	Текст	Число	Текст
1	1	Hello	2	Привет	1	1	Uout=	24,1	В
2	2	Hello World	1	Работает	2	2	Ips=	103,4	А
3	1	Atmel	2	Атмел	1	3	Uд=	31.4	В
4	1	Description	2	АТмега8	1	4	T=	145	°C
5	2	ATmega16	1	Напряжение	2	5	Rси=	18	Ом
6	2	Address	1	Понедельник	2	3	Fosc=	104	Гц
7	1	Counter	2	Институт	2	6	Q=	205	ВАР
8	1	Cursor	2	Суббота	1	2	Pн=	123	Вт
9	2	My name	1	Мое имя	1	4	Z=	103	Ом
10	1	Internal	2	Каникулы	2	2	S=	189	ВАР
11	1	Control	2	Сессия	1	1	T=	203	Сек..

4. Зміст звіту

4.1 Титульний листок із назвою роботи та переліком виконавців.

4.2 Тексти програм.

4.3 Висновки.

5. Контрольні запитання

5.1 Визначити класифікацію та загальні властивості рідкокристалічних індикаторів (РКІ).

5.2 Як використовується вбудований контролер дисплею HD44780?

5.3 Навіщо потрібні виводи RS, R/W, E? Пояснити їх призначення.

5.4 Які схеми інтерфейсів використовуються для підключення РКІ?

5.5 Опишіть структуру та властивості знакогенератора РКІ.

5.6 Опишіть порядок роботи з утилітою HD44780.exe по створенню текстових повідомлень.

5.7 Опишіть процедуру ініціалізації дисплея у разі використання 8-бітового інтерфейсу.

6. Література

6.1 Евстифеев А.В. Микроконтроллеры AVR семейства Classic фирмы “Atmel”. – М.: Издательский дом «Додэка - XXI», 2002.

6.2 Евстифеев А.В. Микроконтроллеры AVR семейства Tiny и Mega фирмы “Atmel”. – М.: Издательский дом «Додэка - XXI», 2004.

6.3 http://atmel.ru/Articles/Atmel17.htm «Сопряжение AVR-микроконтроллеров и ЖКИ».

7. Теоретичні відомості

7.1 **Загальні відомості про рідкокристалічні символьні дисплеї**

Алфавітно-цифрові рідкокристалічні дисплеї (РКІ) є недорогим і зручним рішенням, що дозволяє заощадити час і ресурси при розробці нових виробів, забезпечують відображення великого об'єму інформації.

Такі дисплеї випускаються з елементами підсвічування, що дозволяє їх експлуатувати при зниженій освітленості. Існують вироби з розширеним температурним діапазоном (-20°С...+70°С), які можуть працювати в складних експлуатаційних умовах, зокрема в переносній апаратурі.

Значна кількість алфавітно-цифрових дисплеїв випускається з вбудованим контролером HD44780 фірми Hitachi, або йому сумісним. Контролер забезпечує автоматичну регенерацію зображення на екрані дисплею. Його використання суттєво значно спрощує програми-драйвери індикаторів.

По кількості рядків та знакомісць існує декілька поширених форматів дисплеїв (символів х рядків): 8х2, 16х1, 16х2, 16х4, 20х1, 20х2, 20х4, 24х2, 40х2, 40х4 та менш поширених, таких як 8х1, 12х2, 32х2.

В рамках одного конструктиву РКІ-модуль може мати ряд модифікацій. Зокрема, можуть застосовуватися індикатори, що відрізняються кольором фону і кольором символів. Рідкокристалічні дисплеї можуть оснащуватися заднім

підсвічуванням, яке може бути реалізована декількома способами: за допомогою електролюмінісцентної панелі, люмінісцентної лампи з холодним катодам (CCFL), та на основі світлодіодної матриці (LED).

7.2 **Опис виводів дисплею. Схеми підключення**

В якості прикладу розглянемо дисплей WH1602B фірми Winstar.

У табл..4.2 приведено опис виводів дисплея.

Таблиця 4.2

Контакт	Символ	Рівень	Опис
1	Vss	0В	Нуль напруги живлення
2	Vdd	5,0В	Напруга живлення для логіки
3	Vo	0 - 5В	Регулювання яскравості
4	RS	H/L	H-дані, L-код інструкції
5	RW	H/L	H-читання, L-запис
6	E	H, H/L	Строб передачі даних/команд
7	DB0	H/L	Дані, біт «0»
8	DB1	H/L	Дані, біт «1»
9	DB2	H/L	Дані, біт «2»
10	DB3	H/L	Дані, біт «3»
11	DB4	H/L	Дані, біт «4»
12	DB5	H/L	Дані, біт «5»
13	DB6	H/L	Дані, біт «6»
14	DB7	H/L	Дані, біт «7»
15	A	-	Анод світлодіода підсвічування
16	K	-	Катод світлодіода підсвічування

Звичайно використовують дві схеми підключення дисплею – 4- та 8-бітову. На відміну від 8-бітової схеми у 4-бітовій шина даних формується лише за допомогою ліній даних DB4 - DB7. Шина керування та живлення дисплею реалізуються однаково. На рис. 4.1 приведено 8-бітову схему включення дисплею, яку використано у лабораторному стенді. Ця схема
дозволяє досліджувати як 8- так і 4-бітовий варіанти інтерфейсу.

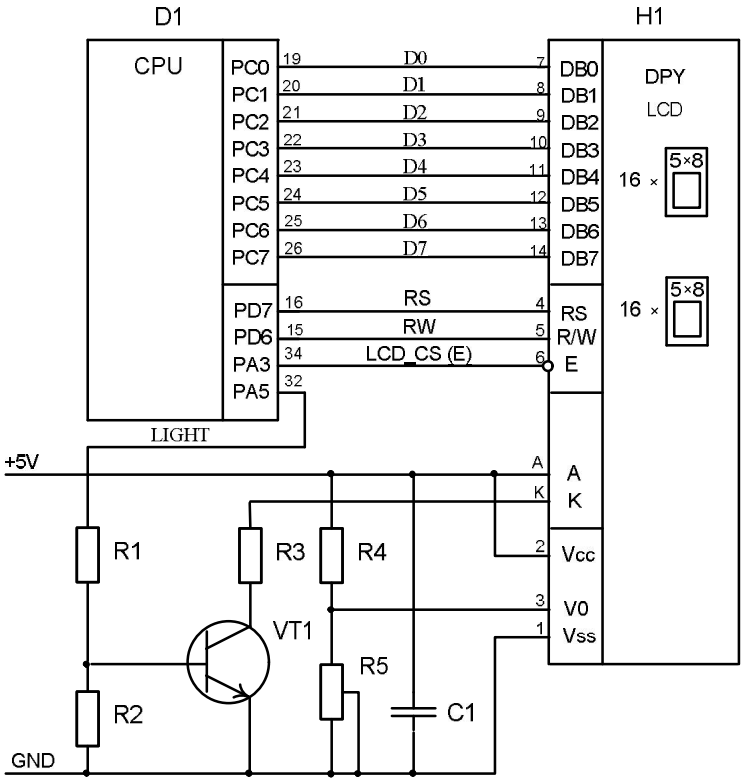


Рис.4.1

7.3 **Загальні відомості про вбудований контролер HD44780**

До складу дисплея входять:

- рідкокристалічна матриця що має 2 рядки по 16 знакомісць. Кожне знакомісце виконано у вигляді матриці що має набір 5x11 пікселей;
- контролер HD44780;
- системи живлення, синхронізації та сканування знакомісць.

До складу контролера входять:

- регістри команд (РК) та даних (РД);
- пам'ять даних, що відображаються - DDRAM;
- пам'ять знакогенератора - CGRAM.

Регістр команд використовується для передачі команд керування дисплеєм та адреси відповідних регістрів DDRAM та CGRAM пам’яті. Під час читання вмісту РК доступна інформація про стан прапора занятості контролера на час виконання чергової команди (BF) та поточний стан рахівника адреси пам’яті (АС). Для доступу до РК на шині керування задається рівень RS = 0.

За допомогою регістру даних проводиться обмін даними з DDRAM або CGRAM областями пам’яті вбудованого контролера. Для доступу до РД на шині керування задається рівень RS=1. Запис інформації забезпечується при стані лінії шини керування RW=0, а читання, відповідно, при RW=1.

Кожна операція обміну даними супроводжується подачею стробу (E). На рис.4.2 зображені часові діаграми, що ілюструють запис інформації.

До пам'яті даних DDRAM (Display Data RAM) записують ASCII - коди символів, що відображаються. Ємність цієї пам’яті складає 80 байт. На рис.4.3 показано співвідношення

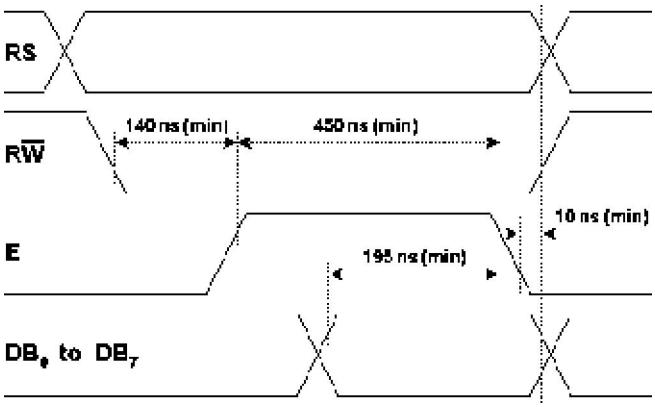


Рис. 4.2

між адресами коду в DDRAM та положенням зображення на дисплеї з двома рядками.

Наявність додаткових адрес DDRAM та можливості зміщувати зображення дозволяє виводити на дисплей рухомі текстові повідомлення.

Пам'ять знакогенератора CGRAM (Character Generator RAM) має дві

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

Рис.4.3

області. У першій області сталої пам’яті (ROM) записані бітові комбінації, які дозволяють відтворювати на знакомісцях певні символи. Друга область RAM – типу. До цієї області користувач може записувати свої власні унікальні графічні зображення. Їх кількість обмежена та залежить від формату зображення. Наприклад, у разі виводу зображень форматом 5x8 пікселей можна сформувати 8 унікальних символів, які розміщують за адресами $00 - $07. У табл..4.3 зображено приклад формування зображення символу «R» у форматі 5x8 пікселей з розміщенням за адресою CGRAM $00. Адреса та дані представлені в бінарному форматі.

Таблиця 4.3

Код символу у DDRAM	Адреса CGRAM (RAM)								Дані CGRAM (RAM)								Зони елемента
	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	
00000000b	*	*	0	0	0	0	0	0	*	*	*	1	1	1	1	0	Зона символу
	*	*	0	0	0	0	0	1	*	*	*	1	0	0	0	1	
	*	*	0	0	0	0	1	0	*	*	*	1	0	0	0	1	
	*	*	0	0	0	0	1	1	*	*	*	1	1	1	1	0	
	*	*	0	0	0	1	0	0	*	*	*	1	0	1	0	0	
	*	*	0	0	0	1	0	1	*	*	*	1	0	0	1	0	
	*	*	0	0	0	1	1	0	*	*	*	1	0	0	0	1	
	*	*	0	0	0	1	1	1	*	*	*	0	0	0	0	0	Зона курсору

У табл. 4.4 приведена відповідність зображень ASCII- кодам русифікованого дисплея для області CGRAM (ROM). У верхній строчці таблиці розміщено старшу (H), а в лівому стовпчику – молодшу тетраду (L) байту коду. Наприклад, ASCII-код цифри «0» - $30.

Таблиця..4.4

H / L	LLLL	LLLH	LLHL	LLHH	LHLL	LHLH	LHHL	LHHH	HLLL	HLLH	HLHL	HLHH	HHLL	HHLH	HHHL	HHHH
LLLL	CG RAM (1)															
LLLH	CG RAM (2)															
LLHL	CG RAM (3)															
LLHH	CG RAM (4)															
LHLL	CG RAM (5)															
LHLH	CG RAM (6)															
LHHL	CG RAM (7)															
LHHH	CG RAM (8)															
HLLL	CG RAM (1)															
HLLH	CG RAM (2)															
HLHL	CG RAM (3)															
HLHH	CG RAM (4)															
HHLL	CG RAM (5)															
HHLH	CG RAM (6)															
HHHL	CG RAM (7)															
HHHH	CG RAM (8)															

7.4 **Система команд контролера HD44780**

У табл.4.5 приведено перелік можливих команд контролера HD44780 (для дисплея з двома рядками), станів шини даних/керування, опис

результатів їх дії та часу виконання (для частоти генератора 270кГц).

Таблиця 4.5

Команда	Код команди										Опис дії команди	Час [мкс]
	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0		
Очищення дисплею	0	0	0	0	0	0	0	0	0	1	Дисплей очищається. Рахівник адреси DDRAM скидається в 0.	1530
Перехід на початок DDRAM	0	0	0	0	0	0	0	0	1	*	Скидається рахівник адреси DDRAM (адреса $0). Вміст DDRAM при цьому не змінюється.	1530
Режим вводу даних	0	0	0	0	0	0	0	1	ID	S	ID=0 – інкремент рахівника адреси DDRAM (CGRAM) після запису; ID=1 – декремент рахівника адреси DDRAM (CGRAM) після запису; S=0 – заборона зсуву тексту дисплея; S=1 – дозвіл зсуву тексту дисплея.	39
Функції дисплею	0	0	0	0	0	0	1	D	C	B	D=0 – виключити дисплей; D=1 – включити дисплей; C=0 – курсор не відображати; C=1 – курсор відображати; B=0 – символу з курсором не мигає; B=1 – символу з курсором мигає.	39
Змішення тексту чи курсору	0	0	0	0	0	1	SC	RL	*	*	SC=0 – зміщення курсору; SC=1 – зміщення тексту; RL=0 – зміщення вліво; RL=1 – зміщення направо.	39
Функція ініціалізації	0	0	0	0	1	DL	N	F	*	*	DL=0 – шина даних 4 біта; DL=1 – шина даних 8 біт; N =0 – дисплей з одним рядком; N =1 – дисплей з двома рядками; F =0 – шрифт 5x7 пікселей; F =1 – шрифт 5x10 пікселей.	39
Адреса CGRAM	0	0	0	1	ACG						Встановлення поточної адреси CGRAM.	39
Адреса DDRAM	0	0	1	ADD							Встановлення поточної адреси DDRAM.	39
Читання флагу BF	0	1	BF	AC							Читання флагу BF і лічильника поточної адреси (AC).	1
Запис даних	1	0	Байт даних								Запис даних у DDRAM або CGRAM.	43
Читання даних	1	1	Байт даних								Читання даних із DDRAM або CGRAM.	43

7.5 **Драйвери індикатора**

Набір драйверів визначається типом інтерфейсу індикатора – 4- чи 8-бітовим. Для кожного типу інтерфейсу необхідно використовувати відповідні підпрограми драйверів.

Набір драйверів індикатора для 8-бітового інтерфейсу приведено у прикладі наскрізного проекту, у файлі драйверів lcd.h.

До набору драйверів індикатора входять три групи функцій.

Першу групу складають функцій, що формують сигнали даних та керування:

- Формування стробу передачі коду в індикатор – «lcd_strobe». Ця функція дозволяє сформувати на шині керування імпульс запису коду з шини даних у відповідні регістри контролера індикатора та відповідну затримку, необхідну для виконання поточної команди;
- Передачі байту на індикатор – «lcd_send». Ця функція дозволяє сформувати послідовність операцій необхідних для передачі у контролер індикатора байту;
- Формування затримки на час виконання команди у контролері індикатора – «Delay». Ця функція дозволяє сформувати затримку на час виконання контролером індикатора поточної команди.

До другої групи входить функція ініціалізації –«InitLCD». Ініціалізація є обов'язковою процедурою, що забезпечує налаштування індикатора на певний режим роботи.

До третьої групи входять функції виводу повідомлень:

- Очистки табло «LCD_Clrscr»;
- Встановлення адреси знакомісця виводу «gotoz»;
- Вивід символу – «OutChar». Ця функція дозволяє виводити певний символ. Для передачі параметрів використовується змінна функції «litera»;
- Вивід повідомлення з 16 символів – «outtext». Для передачі параметрів використовується вказівник на масив;
- Вивід повідомлення з 16 символів у першу або другу строчки, відповідно «DisplayTextL1» та «DisplayTextL2». Для передачі параметрів використовується вказівник на масив;
- Вивід три розрядного числа у певний рядок, починаючи з певного місця «DisplayData». Для передачі параметрів використовується змінна «tmp».

7.6 **Ініціалізація LCD**

Алгоритми ініціалізації відрізняються для 4 та 8-бітного інтерфейсів.

Для 8-бітного інтерфейсу ініціалізацію проводять наступним чином:

- після подачі напруги живлення виконують затримку тривалістю не менше як 15мс, після чого до регістру команд передають команду ініціалізації 0b0011****;
- виконують затримку тривалістю не менш як 4,1мс, після чого до регістру команд ще раз передають ту само команду ініціалізації;
- виконують затримку тривалістю не менш як 100мкс, після чого до регістру команд ще раз передають ту само команду ініціалізації;
- до регістру команд передають команду встановлення функцій 0b0011NF** та виконують затримку тривалістю не менш як 39мкс;

- до регістру команд передають команду вимкнення дисплею 0b00001000 та виконують затримку тривалістю не менш як 39мкс;
- до регістру команд передають команду очистки дисплею 0b00000001 та виконують затримку тривалістю не менш як 39мкс;
- до регістру команд передають команду встановлення режиму 0b000001I/DS та виконують затримку тривалістю не менш як 39мкс.

Під час ініціалізації для формування часових затримок 15мс, 4.1мс та 100мкс не допускається використання біту стану контролера BF. Цей біт можна використовувати лише для наступних операцій керування контролером.

Приклад функції ініціалізації для 8-бітного режиму роботи індикатора «InitLCD» приведено у прикладі наскрізного проекту, у файлі драйверів індикатора lcd.h.

7.7 **Підготовка текстових повідомлень.**

При підготовці текстових повідомлень доцільно використовувати наступні прийоми.

7.7.1 Якщо текст включає лише латинські символи.

Повідомлення доцільно оформляти у вигляді стрінгу.

У цьому випадку вивід стрінгу (наприклад, "Hello World") може бути реалізовано з використанням функції `outtext ("Hello World").`

Слід зауважити, що драйвер виводу текстового повідомлення «OutLine» завжди виводить у строчку по 16 символів. Тому стрінг доцільно доповнювати пробілами до 16 символів.

7.7.2 Якщо виводяться цифри від 0 до 9, причому їх значення може змінюватися, наприклад при формуванні цифрових індикаторів.

У цьому випадку доцільно виводити ASCII- коди цифр із використанням підпрограми виводу окремих символів «outchar». Коди цифр мають зміщення на сталу величину $30. Тому перед виводом цифри код можна отримати шляхом додавання величини зміщення до значення цифри. Наприклад, код цифри «4» - $30+4 =$34. У функції «`DisplayData`» це враховано шляхом додавання до значення цифри
коду «0» – `outchar ('0'+tmp/100).`

7.7.3 Якщо текст включає кирилізовані символи, то можлива ручна побудова масиву, з використанням табл.4.4. Такий спосіб формування повідомлень досить важкий. У цьому випадку доцільно використовувати утиліту генерації текстових повідомлень HD44780.exe, яка розроблена для створення С-програм.

7.7.4 Утиліта підготовки текстових повідомлень HD44780.exe

На рис.4.4 зображено головне вікно програми.

Розглянемо інтерфейс користувача програми. У програмі використовується інтерактивна система підказок. Якщо невідомий якийсь

елемент вікна програми, необхідно навести на нього курсор, що приведе до появи короткої підказки.

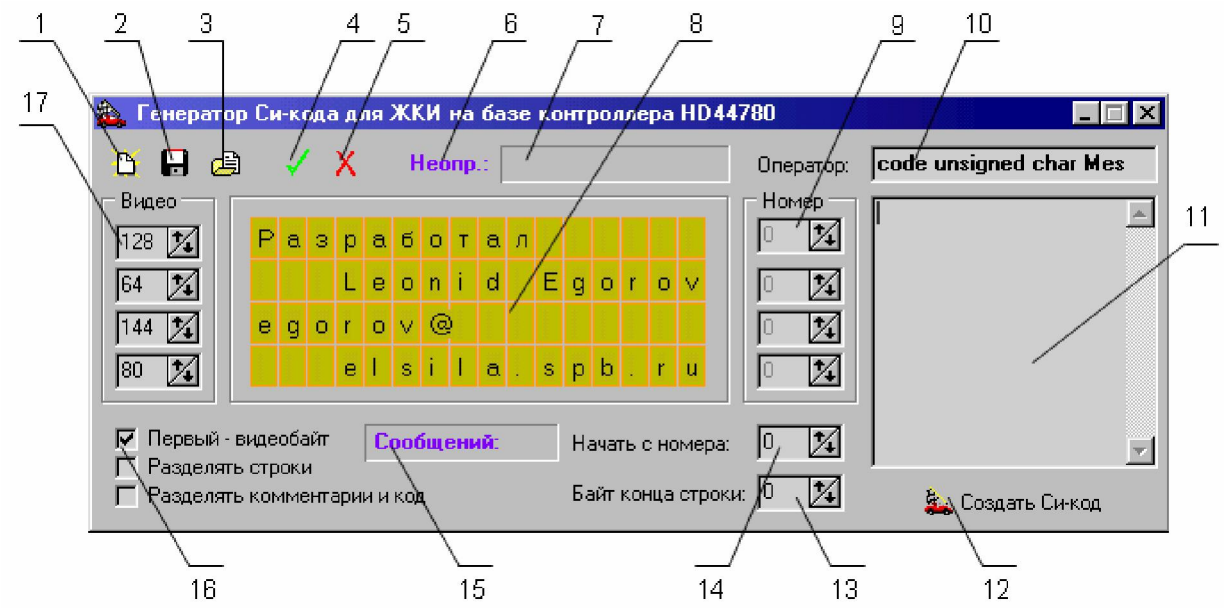


Рис.4.4

У програмі використовуються наступні елементи керування:

1. Кнопка прискореного виклику – «Створення нового проекту».

Очищає всі повідомлення і їх адреси початку у пам'яті програми.

2. Кнопка прискореного виклику – «Зберегти як...».

Зберігає текст або як проект *.lcd (він доступний для подальшого завантаження і правки) або як *.h – заголовочний файл.

3. Кнопка прискореного виклику – «Відкрити проект».

Відкриває проект *.lcd. Відкриваються всі відеообласті (17), положення всіх покажчиків (9), текстовий оператор (10), всі текстові повідомлення проекту (11).

4. Кнопка прискореного виклику – «Показати всі повідомлення».

Використовується при переводі повідомлень (9,8) з рядка в рядок. У протилежному випадку попереднє повідомлення не стирається з екрану.

5. Кнопка прискореного виклику – «Очистити екран».

Очищає дисплей (8), всі відеообласті зберігаються.

6. Індикатор номеру поточного рядка. Якщо для поточного повідомлення, що формується у зоні (11), не задана відеоадреса, відображається слово «Неопр». Для встановлення необхідної адреси необхідно навести курсор на вікно (8) та сформувати клік кнопкою миші.

9. Індикатор номеру повідомлення.

Введіть декілька рядків в поле списку повідомлень, розділяючи їх клавішею Enter, кожній строчці буде присвоєний свій номер. Якщо встановити покажчик (9) на потрібному повідомленні, перевести курсор на потрібне

знакомісце дисплея (8) і клікнути кнопку миші, відбудеться позиціонування повідомлення з цим номером на екрані в задану область.

10. Заголовок повідомлення.

Цей заголовок буде стояти перед кодами кожного сформованого повідомлення. Для приведеного вище фрагмента він виглядав так: «unsigned char Mes». Номер повідомлення MESX генерується автоматично.

11. В цій області формуються тексти повідомлень.

12. Формування кодів у буфері обміну.

Після закінчення введення всіх повідомлень збережіть проект і натисніть клавішу «Створити Сі-код». Код буде розташований в буфері обміну.

13. Байти початку повідомлення (16) та кінця повідомлення (13). Можлива корекція байтів початку повідомлення (17).

Розглянемо приклад формування повідомлення «Команда».

```
/* Date: 31.05.2010   Time: 15:29:46 */
/* Maximum length of a line: 7  byte */
/* In total byte: 8              */
/* FileName: вK                  */
/* [0] "Команда" */  unsigned char Mes0[]={75,111,188,97,189,227,97,0};
```

7.8 **Приклад наскрізного проекту**

У проекті реалізовано вивід повідомлень у перший ("Работа") та другий ("Программирование") рядки індикатора та вивід символу «234» у другий рядок з третього знакомісця.

Проект включає чотири файли Lab_4.c, delay.h, lcd.h, init.h.

Головний файл проекту Lab_4.c має наступний вигляд.

```
//-----------------------------------------------------------
//ICC-AVR application builder : 14.06.2008 1:27:18
//Target : M16
//Crystal: 10.000Mhz
#include <iom16v.h> //файл визначень мікроконтролера ATmega16
#include <macros.h>
//---------- Вкладені файли
#include <delay.h>  //файл програмних затримок
#include <lcd.h>    //файл драйверів індикатора
#include <init.h>   //файл ініціалізації
//----------------------------
//---------- Визначення масивів повідомлень
/* "Работа" */
unsigned char Mes1[]={80,97,178,111,191,97,0};
/* "Программирование" */
unsigned char Mes2[]={168,112,111,180,112,97,188,188,184,112,
111,179,97,189,184,101,0};
//----------------------------

//---------- Головна функція програми
```

```
void main(void)
{
InitDevices();           //Ініціалізація портів та LCD
LCD_LED_ON;              //Вмикання LED - освітлення індикатора
Lcd_Clrscr();            //очистка індикатора
//outtext ("Hello World");    //вивід тексту Hello World
DisplayTextL1(Mes1);     //вивід тексту "Работа" у 1 рядок
DisplayTextL2(Mes2);     //вивід "Программирование" у 2 рядок
DisplayData (234);  //вивід числа 234 у 2 рядок (з 3 місця)
while(1)  {};            //Нескінченний цикл
}
//-------------------------------------------------------------
```

Файл з функціями ініціалізації init.h має наступний вигляд.

```
//-------------------------------------------------------------
//Файл init.h
//-------------------------------------------------------------
//---------- Функція ініціалізації портів
void PortInit(void)
{
PORTA = 0b11001000;          //Початкове значення порту
DDRA  = 0b11111000;          //0- прийом, 1 - передача
PORTB = 0b00000000;
DDRB  = 0b00000000;
PORTC = 0;
DDRC  = 0b11111111;
PORTD = 0b11110010;
DDRD  = 0b11111010;
}
//----------------------------
//---------- Функція ініціалізації мікроконтролера
void InitDevices(void)
{
PortInit();                  //ініціалізація портів
InitLCD ();                  //ініціалізація LCD
}
//-------------------------------------------------------------
```

Файл драйверів індикатора lcd.h має наступний вигляд.

```
//-------------------------------------------------------------//
Файл lcd.h
//-------------------------------------------------------------//--
-------- Символічні імена портів та ліній керування LCD
#define   LCD_PORT_DATA        PORTC
#define   LCD_PORT_DATA_DIR    DDRC
#define   LCD_PORT_DATA_OUT    0xff
#define   LCD_PORT_CONTROL     PORTD
#define   LCD_PORT_CONTROL_DIR DDRD
#define   LCD_RS               (1<<PD7)
#define   LCD_RW               (1<<PD6)
```

```
#define   LCD_E                (1<<PA3)
#define   LCD_LED              (1<<PA5)
//---------- Макрос визначення поточної позиції
#define   gotoxy(x,y)    gotoz((x)|((y)<<6)) //
//---------- Макроси керування лініями LCD
#define   lcd_set_E      (PORTA |= LCD_E)
#define   lcd_set_RS     (LCD_PORT_CONTROL |= LCD_RS)
#define   lcd_set_RW     (LCD_PORT_CONTROL |= LCD_RW)
#define   lcd_clr_E      (PORTA &=~LCD_E)
#define   lcd_clr_RS     (LCD_PORT_CONTROL &=~LCD_RS)
#define   lcd_clr_RW     (LCD_PORT_CONTROL &=~LCD_RW)
#define   LCD_LED_ON     (PORTA |=LCD_LED)
//---------- Команди керування режимами LCD (див. табл.4.5)
#define   LCD_INI         0x30     //ініціалізація
#define   LCD_CLEAR       0x01     //очистка
#define   LCD_HOME        0x02     //перехід на початок екрану
#define   LCD_MODE_SET    0x06     //режим програмування
#define   LCD_OFF         0x08     //вимикання індикатору
#define   LCD_ON          0x0C     //включення індикатору
#define   LCD_NEW_LINE    0xC0     //перехід на нову лінію
#define   LCD_FUNCTION_SET 0x38    //установка функцій
//-------------------------------------------------------------

//---------- Функція формування стробу LCD
void lcd_strobe (void)
{
lcd_set_E;              //встановлення «1» на лінії стробу
DelayMcs(4);            //формування тривалості стробу
lcd_clr_E;              //встановлення «0» на лінії стробу
DelayMcs(4);            //формування затримки після стробу
}
//----------------------------
//---------- Функція передачі байту в LCD
void lcd_send (unsigned char lcddata)
{
LCD_PORT_DATA=lcddata;  //встановлення даних на шині
lcd_strobe();           //формування стробу
DelayMcs(60);           //протокольна затримка після передачі
}
//----------------------------
//---------- Функція очистки LCD
void Lcd_Clrscr (void)
{
lcd_clr_RS;             //встановлення режиму команди
lcd_clr_RW;             //встановлення режиму запису
lcd_send(LCD_CLEAR);    //передача команди очистки дісплею
DelayMs(2);             //протокольна затримка після передачі
}
```

```
//----------------------------
//---------- Функція ініціалізації LCD
void InitLCD (void)
{
LCD_PORT_CONTROL_DIR |=(LCD_RS | LCD_RW); //режим роботи порту
lcd_clr_RS;              //встановлення режиму команди
lcd_clr_RW;              //встановлення режиму запису
DelayMs(20);             //протокольна затримка
lcd_send(LCD_INI);       //передача команди ініціалізації
DelayMs(5);              //протокольна затримка
lcd_send(LCD_INI);       //передача команди ініціалізації
DelayMcs(150);           //протокольна затримка
lcd_send(LCD_INI);       //передача команди ініціалізації
lcd_send(LCD_FUNCTION_SET); //передача команди установки
lcd_send(LCD_OFF);       //передача команди вимикання
lcd_send(LCD_CLEAR);     //передача команди очистки
DelayMs(2);              //протокольна затримка
lcd_send(LCD_MODE_SET);  //передача команди установки режимів
lcd_send(LCD_ON);        // передача команди вмикання
}
//----------------------------
//---------- Функція передачі адреси знакомісця LCD
void gotoz (unsigned char z)
{
lcd_clr_RS;              //встановлення режиму команди
lcd_clr_RW;              //встановлення режиму запису
lcd_send(z | 0x80);      //передача адреси знакомісця
}
//----------------------------
//---------- Функція передачі одного символу
void outchar (unsigned char litera)
{
lcd_clr_RW;              //встановлення режиму запису
lcd_set_RS;              //встановлення режиму даних
lcd_send(litera);        //передача коду символу
}
//----------------------------
//---------- Функція передачі рядка із 16 символів
void outtext (unsigned char *text)
{
unsigned char i;
for (i = 0; text[i] && i<16; i++) outchar (text[i]);
}
//----------------------------
//---------- Функція виводу тексту у 1 рядок
void DisplayTextL1(unsigned char *tmp)
{
gotoxy(0,0);   //встановлення адреси на початку 1 рядка
```

```
outtext(tmp);  //вивід тексту
}
//---------------------------
//---------- Функція виводу тексту у 2 рядок
void DisplayTextL2(unsigned char *tmp)
{
gotoxy(0,1);   //встановлення адреси на початку 2 рядка
outtext(tmp);  //вивід тексту
}
//---------------------------
//---------- Функція виводу три розрядного числа tmp у 2 рядок
void DisplayData(unsigned int tmp)
{
gotoxy(0,1);   //встановлення адреси на початок 2 рядка
outchar ('0'+tmp/100);
outchar ('0'+(tmp%100)/10);
outchar ('0'+(tmp%100)%10);
}
//-------------------------------------------------------------
```

Файл з функціями програмної затримки delay.h має наступний вигляд.

```
//-------------------------------------------------------------
//Файл delay.h
//-------------------------------------------------------------
//--------- Функція програмної затримки на "t_mcs" мікросекунд
void DelayMcs (unsigned int t_mcs)
{while(t_mcs--)
      {asm("nop"); asm("nop"); asm("nop"); asm("nop");};}
//---------------------------
//--------- Функція програмної затримки на "t_ms" мілісекунд
void DelayMs (unsigned int t_ms)
{while(t_ms--) {DelayMcs(1000);};}
//---------------------------
//--------- Функція програмної затримки на "t_s" секунд
void DelayS (unsigned int t_s)
{while(t_s--) {DelayMs(1000);};}
//-------------------------------------------------------------
```

КОМП'ЮТЕРНИЙ ПРАКТИКУМ №5.
КЛАВІАТУРА МІКРОКОНТРОЛЕРА

1. Мета роботи:

- ознайомлення з схемотехнічними прийомами побудови клавіатур;
- засвоєння принципів побудови драйверів клавіатури;
- розробка тестового проекту.

2. Програма роботи:

2.1 Вивчити схемотехнічні прийоми побудови клавіатур;

2.2 Вивчити алгоритми обробки натискання кнопок клавіатури;

2.3 Розробити драйвер клавіатури, що забезпечує ввід інформації у мікроконтролер та задані реакції на натискання кнопок;

2.4 Розробити тестовий проект та перевірити його за допомогою лабораторного стенду.

Увага! Перше програмування провести у присутності викладача;

2.5 Зробити висновки по роботі.

3. Завдання до лабораторної роботи

У табл. 5.1 розташовані варіанти вихідних даних.

Таблиця 5.1

№	Індикатор	Алгоритм вводу	Фіксація натисненого стану	Цикл опиту [сек.]	Час тремтіння [мс]	Реакції на натискання кнопок
1	LCD	A1	ФН	1	1	Узгоджується із викладачем
2	LED	A2	ФН	0,5	2	
3	LCD	A3	ФНВ	0,7	3	
4	LED	A2	ФН	2	2	
5	LCD	A3	ФНВ	0,3	1,3	
6	LED	A1	ФН	0,8	1,8	
7	LCD	A1	ФН	1	3,3	
8	LED	A2	ФНВ	0,4	2,1	
9	LCD	A1	ФНВ	0,2	2,7	
10	LED	A3	ФН	0,9	3	
11	LCD	A2	ФН	1,4	2	
12	LED	A1	ФНВ	1,2	1	

Де:

- A1, A2, A3 – алгоритми вводу стану кнопки із урахуванням тремтіння контактів (A1 із двома опитами стану кнопки та затримкою на час тремтіння контактів; A2 із багаторазовими опитами та мажоритарним принципом розпізнавання стану; A3 із заданим числом співпадаючих значень при багаторазових опитах);
- ФН – фіксація стану при натисканні кнопки;

- ФНВ – фіксація стану при натисканні та наступному відпусканні кнопки.

У стенді використовуються наступні елементи керування:

- Лінія вводу стану кнопки S1 – PB1 (L-активна);
- Лінія вводу стану кнопки S2 – PB3 (L-активна);
- Лінія вводу стану кнопки S3 – PB4 (L-активна).

4. Зміст звіту

4.1 Титульний листок із назвою роботи та переліком виконавців.

4.2 Текст програми.

4.3 Висновки.

5. Контрольні запитання

5.1 Приведіть класифікацію типів клавіатур.

5.2 Опишіть схемотехнічні прийоми використання окремих кнопок.

5.3 Визначте схемотехнічні прийоми побудови клавіатур.

5.4 Опишіть алгоритми ідентифікації натискання кнопок.

5.5 Які алгоритми фіксації натискання кнопок використовують у драйверах клавіатур?

5.6 Які вихідні дані враховують при побудові драйвера клавіатури та підпрограм реакцій на натискання кнопок?

6. Література

6.1 Евстифеев А.В. Микроконтроллеры AVR семейства Classic фирмы “Atmel”. – М.: Издательский дом «Додэка - XXI», 2002.

6.2 Евстифеев А.В. Микроконтроллеры AVR семейства Tiny и Mega фирмы “Atmel”. – М.: Издательский дом «Додэка - XXI», 2004.

7. Теоретичні відомості

7.1. Класифікація клавіатур

При побудові клавіатур, у залежності від їх функціонального призначення, використовують різну кількість кнопок. Розрізняють клавіатури, що призначені:

- Для вводу та редагування тексту (персональні комп’ютери);
- Переважного вводу цифрової інформації, з можливістю вводу тексту (мобільні телефони);
- вводу лише цифрової інформації (калькулятори);
- епізодичного вводу параметрів мікроконтролерних пристроїв (побутова техніка, мікроконтролерні пристрої промислового призначення).

У клавіатурах із інтенсивним використанням кількість кнопок підвищено. Наприклад, розширена клавіатура персонального комп’ютера має понад 100 кнопок.

Із зменшенням інтенсивності використання клавіатури кількість

кнопок зменшується. При цьому для збереження функціональних можливостей клавіатури кнопкам призначаються альтернативні функції та вводять можливості для визначення таких функцій (мобільні телефони).

Типова клавіатура мікроконтролера, що використовується епізодично для налаштування параметрів, має 4-6 кнопок (НВЧ - пічки, пральні машини, цифрові фотоапарати).

7.2. Схемотехнічні прийоми використання окремих кнопок

При розробці принципової схеми клавіатури необхідно вирішувати дві задачі:

- Забезпечення вводу інформації від кнопок;
- Узгодження клавіатури заданого об`єму з мікроконтролером.

Кнопка клавіатури має контактну пару, що комутується. При визначенні режиму роботи контактної пари враховують величину мінімального струму. Для тактових кнопок, які звичайно використовуються при побудові клавіатур, цей струм складає 0,3 – 1 мА. У разі встановлення меншого робочого струму можливі відкази функціонування кнопок.

Для формування цифрових логічних рівній напруги та забезпечення мінімального струму контактів досить часто використовуються схеми, що приведені на рис. 5.1.

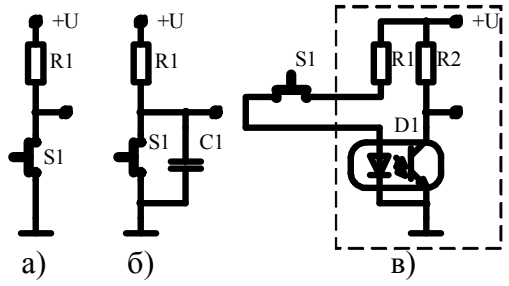


Рис. 5.1

На вибір схемотехнічного рішення суттєво впливають віддаленість кнопки від мікроконтролера та рівень високочастотних завад. При низькому рівні завад та невеликій відстані від кнопки до портів мікроконтролера (0,1-0,2м) використовується схема, що приведена на рис.5.1,а.

При підвищеному рівні завад та відстані від кнопки до мікроконтролера до 1-3 м використовують додатковий конденсатор (схема на рис.5.1,б).

У разі підвищення рівня завад збільшують струм контактної пари до 3 -10 мА та величину ємності конденсатора. У разі необхідності збільшення довжини лінії зв’язку між кнопкою та мікроконтролером більше 5 - 10м доцільно використовувати струмову петлю (10-100мА) та оптичні розв’язки. Таку схему приведено на рис. 5.1,в. Канал оптичної розв’язки дозволяє усунути вплив падіння напруги на лінії зв’язку на рівні напруги, що формуються при включеному та виключеному станах кнопки.

7.3. Схемотехнічні прийоми побудови клавіатур

Розробку принципової схеми клавіатури проводять по заданому числу кнопок. При малому числі кнопок (1-10шт.) звичайно використовують безпосереднє підключення до ліній портів. Такий спосіб побудови клавіатури проілюстровано на рис. 5.2. Якщо число кнопок клавіатури знаходиться у діапазоні від 10 до 30 шт., то для вводу інформації доцільно використовувати додаткові регістри. Принципову схему такої клавіатури приведено на рис. 5.3. У цій схемі для читання інформації з регістру D2 використано паралельну шину, яка сформована на базі порту PA мікроконтролера.

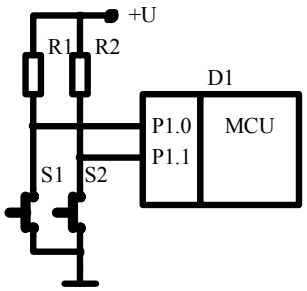


Рис. 5.2

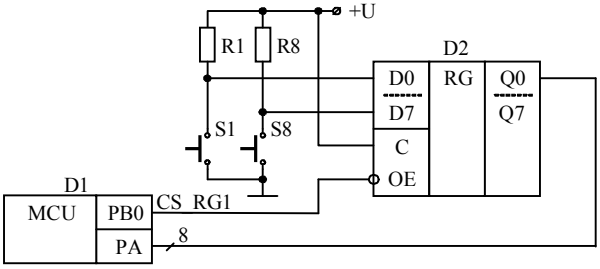


Рис. 5.3

Для стробування додаткових регістрів використовуються лінії порту PB. Таке схемотехнічне рішення дозволяє за допомогою двох портів мікроконтролера та 8 додаткових регістрів провести опитування клавіатури із 64 кнопками.

При розробці клавіатур із числом кнопок, що перевищує 30шт., доцільно використовувати схеми із матричною структурою.

На рис. 5.4 приведено принципову схему, що ілюструє можливість побудови такої клавіатури.

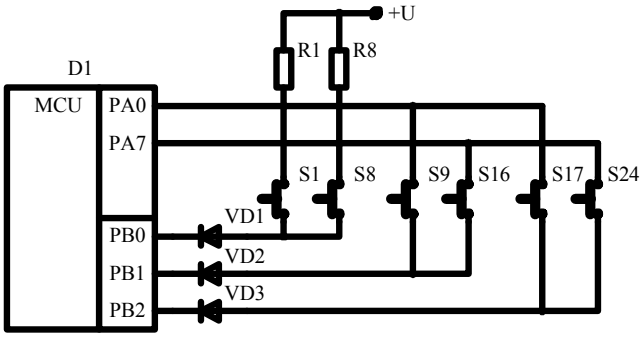


Рис. 5.4

Схема має три групи по 8 кнопок S1-S8, S9-S16, S17-S24, які підключено до загальної паралельної шини вводу даних (порт PA). Групи кнопок через діоди VD1-VD3 підключено до ліній сканування PB0 – PB2. Діоди виключають можливість короткого замикання по лініям порту PB, у разі одночасного натискання кнопок різних груп.

На лініях сканування по черзі встановлюється “0”- рівень та проводиться опит стану кнопок активізованої групи. У разі використання двох портів максимальна кількість кнопок такої матричної клавіатури досягає 64. Ця схема більш проста, у порівнянні з попереднім варіантом із додатковими регістрами.

7.4. Алгоритми ідентифікації натискання кнопки

Комутація контактної пари кнопки характеризується тремтінням контактів – багаторазовими перемиканнями із частотою механічного резонансу. Звичайно тремтіння контактів характерно для замикання контактів. Час тремтіння залежить від типу контактів та може складати від 1 до 20мс. У мікроконтролерних системах при побудові клавіатур часто
використовують тактові кнопки із часом тремтіння 1-3мс.

Для виключення впливу багаторазового перемикання контактів на інтервалі тремтіння, необхідно формувати алгоритми ідентифікації стану кнопки з врахуванням цього явища.

Звичайно використовують три типи алгоритмів ідентифікації:

- Із двома опитами та затримкою на час тремтіння контактів;
- Із багаторазовими опитами та мажоритарним принципом розпізнавання стану;
- Із заданим числом співпадаючих значень, при багаторазових опитах.

При використанні першого алгоритму проводиться періодичне опитування стану кнопок клавіатури. У разі реєстрації натисненого стану кнопки, для виключення впливу тремтіння контактів, проводиться затримка на час тремтіння, потім – повторне опитування. Результат повторного опитування відповідає стану кнопки.

У другому алгоритмі проводяться багаторазові опитування стану кнопки. Визначаються числа результатів експериментів, що відповідають різним станам кнопки. На підставі порівняння цих чисел, по мажоритарному принципу приймається рішення про стан кнопки. Для виключення впливу тремтіння контактів число опитувань обирають таким чином, щоб час використаний на багаторазові опитування перевищував час тремтіння контактів.

При реалізації третього алгоритму проводиться задане число експериментів. Якщо у серії опитувань отримані різні результати про стан кнопки, то результати анулюються, а серія повторюється. Як і в попередньому алгоритмі кількість опитувань обирають у відповідності із часом тремтіння контактної пари.

7.5. Драйвер клавіатури

При написанні драйвера клавіатури задаються:

- числом кнопок;
- часом тремтіння контактів та типом алгоритму вилучення його впливу;
- часом циклу опитування стану кнопок;
- способом реєстрації натискання кнопки.

Час циклу опитування стану кнопок визначає швидкодію клавіатури. Уведення періоду опитування необхідно для узгодження швидкості роботи

мікроконтролерної системи та часу реакції оператора. Звичайно період опитування обирають у діапазоні 0,3 – 1сек.

Для забезпечення заданої циклічності опитування клавіатури, у залежності від особливостей функціональних властивостей мікроконтролера, використовують два прийоми. Якщо у мікроконтролерній системі часові затримки у 0,3 - 1сек при виконанні головного циклу програми є несуттєвими, то вказану затримку реалізують у головному циклі. В іншому випадку циклічне опитування клавіатури забезпечують за рахунок використання переривань від таймера.

Для фіксації натискання кнопок використовують два алгоритми:

- подію реєструють у натисненому стані кнопки;
- подію реєструють у разі, якщо кнопка була натиснена і потім відпущена.

При використанні драйверів, в яких використовується перший алгоритм фіксації, тривале утримання кнопки в натисненому стані приводить до циклічного виконання відповідної підпрограми реакції. Такий драйвер доцільно використовувати у разі необхідності зміни значень параметрів у широкому діапазоні.

Якщо необхідно забезпечити більш чітку модифікацію змінних, із жорсткою фіксацією їх значень по кожному натисканню кнопок, то доцільно використовувати другий алгоритм.

7.5.1. Приклад драйвера клавіатури

Розглянемо приклад драйвера із наступними вихідними даними:

- число кнопок – 3;
- час тремтіння контактів – 3 мс;
- алгоритм вилучення впливу тремтіння контактів – із двома опитуваннями та затримкою на час тремтіння контактів;
- спосіб та час циклічного опитування стану кнопок – підпрограми опитування кнопок та затримки (узгодження з оператором) знаходяться у головному циклі. Час затримки - 0,3 с;
- спосіб реєстрації стану кнопок – при натисканні.

Драйвер реалізує функції налаштування значення двох змінних :

- «Voltage» може мінятися у діапазоні «5 – 100» з кроком «5»,
- «Current» може мінятися у діапазоні «10 – 100» з кроком «10»;
- Перехід від однієї змінною до іншої відбувається при натисканні кнопки «MODE» - (S3). При цьому на LCD - індикатор виводяться текстові повідомлення, відповідно, «Voltage» та «Current»;
- Збільшення значення змінної відбувається при натисканні кнопки «UP» - (S1). Значення змінної виводяться на LCD - індикатор;

- Зменшення значення змінної відбувається при натисканні кнопки «DOWN» - (S2). Значення змінної виводяться на LCD – індикатор.

Проект реалізовано з використанням попередньої лабораторної роботи – по виводу інформації на LCD – індикатор. Проект включає п’ять файлів Lab_5.c, delay.h, lcd.h, init.h та key.h. Зміни відбулися в головному файлі проекту Lab_5.c (включено замість Lab_4.c). Додатково включено файл функцій драйверів клавіатури key.h.

Приклад 1

Головний файл проекту Lab_5.c має наступний вигляд.

У фрагменті ініціалізації проводиться завантаження дистрибутивних значень змінних, та активізація режиму роботи «VOLTAGE».

```
//-----------------------------------------------------------
//ICC-AVR application builder : 14.02.2019 11:34:27
//Target : M16
//Crystal: 10.000Mhz

#include <iom16v.h>      //файл визначень ATmega16
#include <macros.h>
//---------- Вкладені файли
#include <delay.h>       //файл програмних затримок
#include <lcd.h>         //файл драйверів індикатора
#include <init.h>        //файл ініціалізації
#include <key.h>         //файл драйверів клавіатури
//----------------------------
//---------- Головна функція програми
unsigned char Voltage, Current, Regime;

void main(void)
{
InitDevices();           //Ініціалізація портів та LCD
LCD_LED_ON;              //Вмикання LED - освітлення індикатора
//---------- Встановлення дистрибутивного режиму роботи
Regime =VOLTAGE;         //Встановлення дистрибутивного режиму
Voltage=Voltage_MIN;     //Встановлення дистрибутивних значень
Current=Current_MIN;     //змінних
DisplayTextL1("Voltage");//вивід тексту "Voltage" у 1 рядок
DisplayData(Voltage);    //вивід значення змінної "Voltage"
//----------
while(1)
      {
      key();             //виклик функції драйвера клавіатури
      DelayMs(300);      //затримка узгодження з оператором
      };                 //Нескінченний цикл
}
//-----------------------------------------------------------
```

Файл з функціями драйверів клавіатури key.h має наступний вигляд.

```
//---------- Символічні визначення кнопок
#define UP    (1<<PB3)   //визначення лінії кнопки UP
#define DOWN  (1<<PB1)   //визначення лінії кнопки DOWN
#define MODE  (1<<PB4)   //визначення лінії кнопки MODE
//---------- Символічні визначення констант
extern unsigned char Regime;  //змінна поточного режиму
#define VOLTAGE     0         //значення змінної для режиму "U"
#define CURRENT     1         //значення змінної для режиму "I"
//----------
#define   Voltage_MIN    5    //мінімальне  значення «Voltage»
#define   Voltage_MAX    100  //максимальне значення «Voltage»
#define   Voltage_DELTA  5    //величина прирощення  «Voltage»

extern unsigned char Voltage; //змінна режиму «Voltage»
//----------
#define   Current_MIN    10   //мінімальне  значення «Current»
#define   Current_MAX    100  //максимальне значення «Current»
#define   Current_DELTA  10   //величина прирощення  «Current»
extern unsigned char Current; //змінна режиму «Current»
//----------------------------
//---------- Функція опитування клавіатури "UP"
void Key_Up(void)
{
switch (Regime)                      //розгалуження по змінним
     {
     case VOLTAGE:                        //режим VOLTAGE
          {
          if (Voltage < Voltage_MAX)    //перевірка максимуму
             {
              Voltage+=Voltage_DELTA;   //модифікація змінної
              DisplayData(Voltage);     //відображення значення
              };
           }; break;
     case CURRENT:                        //режим CURRENT
               {
               if (Current < Current_MAX)//перевірка максимуму
                  {
               Current+=Current_DELTA;   //модифікація змінної
               DisplayData(Current);     //відображення значення
               };
          }; break;
      };
}
//----------------------------
//---------- Функція опитування клавіатури "DOWN"
void Key_Down(void)
```

```
{
switch (Regime)                    //розгалуження по змінним
     {
     case VOLTAGE:                      //режим VOLTAGE
          {
          if (Voltage > Voltage_MIN)    //перевірка мінімуму
             {
              Voltage -= Voltage_DELTA; //модифікація змінної
              DisplayData(Voltage);     //відображення значення
             };
          }; break;
     case CURRENT:                      //режим CURRENT
          {
          if (Current > Current_MIN)    //перевірка мінімуму
             {
              Current -= Current_DELTA; //модифікація змінної
              DisplayData(Current);     //відображення значення
             };
          }; break;
     };
}
//----------------------------
//---------- Функція опитування клавіатури "MODE"
void Key_Mode(void)
{
switch (Regime)               //розгалуження по змінним
     {
     case VOLTAGE:                 //режим VOLTAGE, перехід
          {                        //режим CURRENT
          Regime =CURRENT;         //модифікація змінної режиму
          Lcd_Clrscr();            //очистка індикатора
          DisplayTextL1("Current");//вивід тексту "Current"
       DisplayData(Current);   //відображення значення
          }; break;
     case CURRENT:                 //режим CURRENT, перехід
               {                   //режим VOLTAGE
               Regime =VOLTAGE;    //модифікація змінної режиму
               Lcd_Clrscr();            //очистка індикатора
               DisplayTextL1("Voltage");//вивід тексту"Voltage"
          DisplayData(Voltage);//відображення значення
               }; break;
     };
}
//----------------------------
//---------- Функція драйвера клавіатури
void key(void)
{
if (!(PINB & UP) | !(PINB & DOWN) | !(PINB & MODE) )//1опитув.
```

```
    {
    DelayMs(3);//затримка 3 мсек,фільтрація тремтіння контактів
    if (!(PINB & UP))     {Key_Up();};   //2 опитування кнопок
    if (!(PINB & DOWN))   {Key_Down();}; //та реакції
    if (!(PINB & MODE))   {Key_Mode();};
    };
}
//---------------------------
```

7.5.2. Головна функція драйвера та функції реакцій на натискання кнопок

Головна функція драйвера **void** key(**void**) реалізує опитування клавіатури та виклик відповідних функцій реакцій на натискання кнопок. У функції реалізовано алгоритм фільтрації тремтіння контактів кнопок з подвійним опитуванням та затримкою на прогнозований час тремтіння.

Зміст функцій реакцій визначається функціональними особливостями клавіатури. У розглянутому вище прикладі драйвера використано три кнопки “MODE”, “UP”, “DOWN”. Кнопка “MODE” використана для зміни режиму роботи. Кнопки “UP”, “DOWN” застосовані для зменшення та збільшення значень змінних.

Функція **void** Key_Mode(**void**) забезпечує перехід від однієї змінної до іншої. У прикладі реалізована селекція двох режимів роботи «VOLTAGE» та «CURRENT». Для визначення стану програми використано змінну режиму «Regime». Для відображення стану програми на LCD-індикатор виводяться повідомлення, «Voltage» або «Current».

У кожному із режимів можлива модифікація однієї із двох змінних із заданими кроком та діапазоном можливих значень. Дистрибутивні значення цих змінних задаються в описовій частині програми.

Функції реакції на натискання кнопок **void** Key_Down(**void**) та **void** Key_Up(**void**) використовують спільний алгоритм. У функціях проводиться розгалуження згідно з активним режимом. Потім проводиться перевірка досягнення меж границі діапазону змінної. Якщо змінна знаходиться у межах діапазону, проводиться модифікація її значення.

При визначенні початкового значення змінної, верхнього та нижнього значень меж діапазону та величини прирощення необхідно враховувати те що ці величини пов’язані між собою.

КОМП'ЮТЕРНИЙ ПРАКТИКУМ № 6.
РОБОТА З КОМПАРАТОРОМ ТА АНАЛОГОВО-ЦИФРОВИМ ПЕРЕТВОРЮВАЧЕМ МІКРОКОНТРОЛЕРА AVR Mega16

1. Мета роботи:

- ознайомлення з методами вводу аналогових сигналів за допомогою вбудованих аналогово-цифрового перетворювача (АЦП) та компаратора мікроконтролера AVR Mega16;
- ознайомлення з методами цифрової фільтрації флуктуацій напруги, що виміряна АЦП;
- розробка тестових проектів.

2. Програма роботи:

2.1 Ознайомитися з методами вводу аналогових сигналів за допомогою вбудованого компаратора.

- Вивчити побудову компаратора, та ознайомитися з відповідними регістрами керування;
- Засвоїти процедуру ініціалізації компаратора;
- З використанням мови Сі скласти програму вводу даних та передачі результату на дискретний індикатор.

2.2 Ознайомитися з методами вводу аналогових сигналів за допомогою вбудованого АЦП.

- Вивчити побудову АЦП, та ознайомитися з відповідними регістрами керування;
- Засвоїти процедуру ініціалізації АЦП;
- Скласти програму вводу даних, порівняння з приведеним у завданні значенням та передачі результату порівняння на дискретний індикатор.

2.3 Ознайомитися з методами цифрової фільтрації флуктуацій напруги. Вивести виміряне значення напруги на LED або LCD індикатор з використанням програмної фільтрації та без неї. Під час виконання завдання використати програми із робіт по виводу даних на індикатор.

2.4 Перевірити розроблені проекти за допомогою стенду.

Увага! Перше програмування провести у присутності викладача;

2.5 Зробити висновки по роботі.

3. Завдання до лабораторної роботи

У таблиці 6.1 розташовані варіанти вихідних даних.

Для всіх варіантів завдань неінвертуючий вхід компаратора приєднано до вбудованого джерела опорної напруги, а інвертуючий – до входу ADC0 аналогово-цифрового перетворювача.

Таблиця 6.1

№	П.2.1		П.2.2		П.2.3		
	Сигнал переривання	Реакція	Uin [В]	Реакція	Алгоритм	N	Індикатор
1	OT	LED1	1,4	BEEP	MAV	4	LCD
2	FE	LED2	0,5	LIGHT	Σ/N	8	LED
3	RE	BEEP	2,0	LED1	MAV	8	LCD
4	OT	LIGHT	1,1	LED2	Σ/N	16	LED
5	FE	LED1	0,4	BEEP	Σ/N	4	LCD
6	RE	LED2	0,2	LED2	Σ/N	2	LED
7	FE	BEEP	1,8	LED1	MAV	16	LED
8	RE	LIGHT	1,3	LED2	MAV	2	LCD
9	OT	LED1	1,9	BEEP	Σ/N	8	LED
10	RE	LED2	2,0	LIGHT	Σ/N	16	LED
11	OT	BEEP	2,2	LED1	MAV	32	LCD
12	RE	LED2	1,2	LED2	Σ/N	32	LED

Де:

- OT - переривання при будь якій переміні вихідного стану;
- FE - переривання при падаючому фронті вихідного сигналу;
- RE - переривання при наростаючому фронті вихідного сигналу;
- Σ/N - алгоритм цифрової фільтрації результатів вимірювань з простим усереднюванням;
- MAV - алгоритм цифрової фільтрації результатів вимірювань з ковзаючим усереднюванням;
- N – число вимірювань;
- LED та LCD – типи індикаторів.

У стенді використовуються наступні елементи керування :

- Лінія вводу аналогового сигналу АЦП – PA0;
- Лінія вводу аналогового сигналу компаратора PD2-підключена до джерела синхроімпульсів частоти мережі живлення;
- Лінія керування світлодіодом LED1 (PA6, L - активна);
- Лінія керування світлодіодом LED2 (PA7, L - активна);
- Лінія керування світлодіодом LCD (PA5, H - активна);
- Лінія керування гудком BEEP (PA4, H - активна).

4. Зміст звіту

4.1 Титульний листок із назвою роботи та переліком виконавців.

4.2 Тексти програм.

4.3 Висновки.

5. Контрольні запитання

5.1 Визначити варіанти ініціалізації входів компаратора.

5.2 Як задається тип реакції компаратора на зміну вхідної напруги?

5.3 Як задається тип джерела опорної напруги АЦП?

5.4 Визначити методи запуску перетворювань АЦП.

5.5 Як задається час перетворення АЦП?

5.6 Визначити варіанти ініціалізації входів АЦП.

6. Література

6.1 Евстифеев А.В. Микроконтроллеры AVR семейства Classic фирмы “Atmel”. – М.: Издательский дом «Додэка - XXI», 2002.

6.2 Евстифеев А.В. Микроконтроллеры AVR семейства Tiny и Mega фирмы “Atmel”. – М.: Издательский дом «Додэка - XXI», 2004.

7. Теоретичні відомості

7.1 **Аналоговий компаратор**

Модуль аналогового компаратора включено до складу всіх Mega AVR мікроконтролерів. Структурну схема модуля компаратора наведено на рис.6.1.

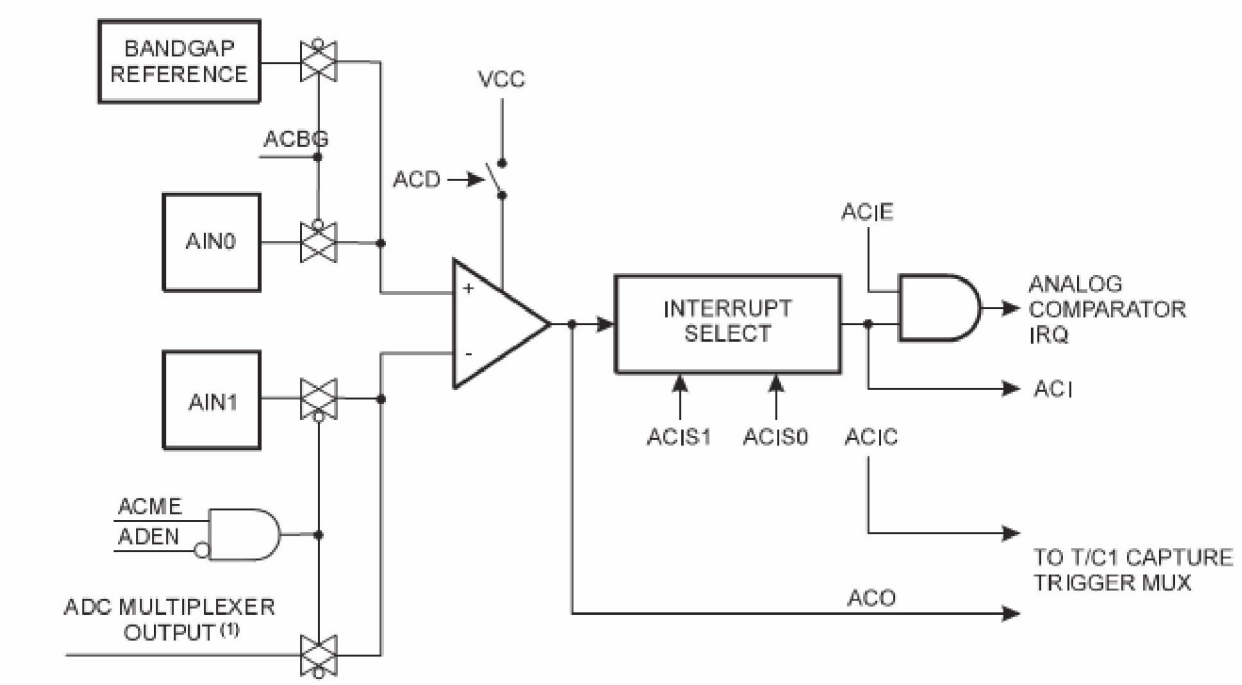


Рис.6.1. Структурна схема модуля компаратора

Компаратор дозволяє порівнювати значення аналогової напруги на інвертуючому та неінвертуючому входах. Існує декілька варіантів комутації цих виводів.

Неінвертуючий вхід компаратора може бути підключеним до виводу мікроконтролера AIN0 або до внутрішнього джерела опорної напруги (1,22 В).

Інвертуючий вхід компаратора може бути підключеним до виводу AIN1 мікроконтролера, або будь якого вхідного виводу аналогово-цифрового перетворювача – ADC0-ADC7. Для використання вхідних виводів мікроконтролера для вводу аналогової напруги необхідно налаштувати

відповідні регістри DDRx на режим приймача, а у регістрах PORTx відключити внутрішні опори підтяжки.

Для реєстрації вихідного стану компаратора можна використовувати переривання, або біт вихідного стану компаратора AC0 (регістр ACSR).

7.1.1 Регістри керування компаратором

Для керування режимами роботи компаратора використовують регістри ACSR та SFIOR.

Bit	7	6	5	4	3	2	1	0	
	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	ACSR
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	N/A	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	SFIOR
Read/Write	R/W	R/W	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Біти регістру ACSR мають наступне призначення («1» - активна):

- ACD - заборона компаратора;
- ACBG - підключення неінвертуючого входу компаратора до джерела опорної напруги;
- AC0 - вихід компаратора;
- ACI - прапор переривань компаратора;
- ACIE - дозвіл переривань від компаратора;
- ACIC - підключення компаратора до схеми захвату таймера TC1;
- ACIS1, ACIS0 – визначення типу сигналу, що викликає переривання. Нижче приведено відповідність режимів роботи системи переривань від стану бітів ACIS1, ACIS0.

ACIS1	ACIS0	Interrupt Mode
0	0	Comparator Interrupt on Output Toggle
0	1	Reserved
1	0	Comparator Interrupt on Falling Output Edge
1	1	Comparator Interrupt on Rising Output Edge

У регістрі SFIOR для керування режимами роботи компаратора використовується лише один біт ACME. У залежності від стану цього біту до неінвертуючого входу компаратора підключаються вивід AIN1 мікроконтролера, або вхідні виводи аналогово-цифрового перетворювача, які визначаються регістром ADMUX.

ACME	ADEN	MUX2..0	Analog Comparator Negative Input
0	x	xxx	AIN1
1	1	xxx	AIN1
1	0	000	ADC0
1	0	001	ADC1
1	0	010	ADC2
1	0	011	ADC3
1	0	100	ADC4
1	0	101	ADC5
1	0	110	ADC6
1	0	111	ADC7

Важливу роль при цьому відіграє також стан біту ADEN регістру ADCSRA. Аналогово-цифровий перетворювач необхідно відключити (ADEN = 0).

7.1.2 Приклад використання компаратора

У першому прикладі приведено програму, де компаратор використовує для порівняння напругу вбудованого джерела та напругу, що заводиться зі змінного резистора стенду на нульовий канал АЦП. Реалізується принцип генерації переривань по будь-якому фронту на виході компаратора (режим ОТ). У підпрограмі переривань проводиться аналіз вихідного стану компаратора (біт ACO регістру ACSR) та у залежності від цього формується сигнал керування світлодіодом LED1.

При ініціалізації компаратора були використані наступні налаштування:

- Компаратор включено (ACD=0) ;
- Інвертуючий вхід підключено до внутрішнього джерела опорної напруги (ACBG=1);
- Неінвертуючий вхід підключено до виводу мікроконтролера AIN1;
- Дозволені переривання від компаратора (ACIE=1);
- Переривання виникають у разі будь якої зміни напруги на виході компаратора (ACIS0=ACIS1=0).

Приклад 1

```
//----------------------------
//ICC-AVR application builder : 11.12.2019 11:12:33
//Target : M16
//Crystal: 10.000Mhz

#include <iom16v.h>      //файл визначень ATmega16
#include <macros.h>
//---------- Визначення ліній портів
#define LED1 6           //лінія підключення світлодіоду LED1
```

```
//----------------------------
//---------- Функція ініціалізації портів
void port_init(void)
{
PORTA = 0b01000000;      //початковий стан LED1 - виключено
DDRA  = 0b01000000;      //лінія керування LED1 – передавач
              //вхід АЦП (PA0) – приймач, підтяжка відсутня
}
//----------------------------
//---------- Функція ініціалізації компаратора
void comparator_init(void)
{
ACSR = (1<<ACBG) | (1<<ACIE); //підключено внутрішню Uref,
                              //дозволені переривання
SFIOR  = (1<<ACME);           //дозволена робота з каналами АЦП
ADCSRA = 0x00;                //АЦП виключено
ADMUX  = 0x00; //підключено до входу компаратора лінію АЦП ADC0
}
//----------------------------
//---------- Функція ініціалізація мікроконтролера
void init_devices(void)
{
CLI();                        //заборона переривань
port_init();                  //ініціалізація портів
comparator_init();            //ініціалізація компаратора
SEI();                        //дозвіл переривань
}
//----------------------------
//---------- Функція переривань компаратора
#pragma interrupt_handler ana_comp_isr:17
void ana_comp_isr(void)
{
if     (ACSR & (1<<ACO))      //перевірка вихідного стану
      {PORTA |= (1<<LED1);}   //вимикання світлодіоду LED1
else
      {PORTA &= ~(1<<LED1);}; //включення світлодіоду LED1
}
//----------------------------
//---------- Головна Функція програми
void main(void)
{
init_devices();               //ініціалізація мікроконтролера
while(1)                      //нескінченний цикл
{};
}
//----------------------------
```

7.2 **Аналогово-цифровий перетворювач**

Ряд моделей мікроконтролерів сімейства мають в своєму складі багатоканальний 10-розрядний АЦП послідовного наближення. Число каналів залежить від моделі. В якості входів модуля АЦП в моделі ATmega16 використовуються виводи порту А.

Структурну схему модуля АЦП наведено на рис.6.2.

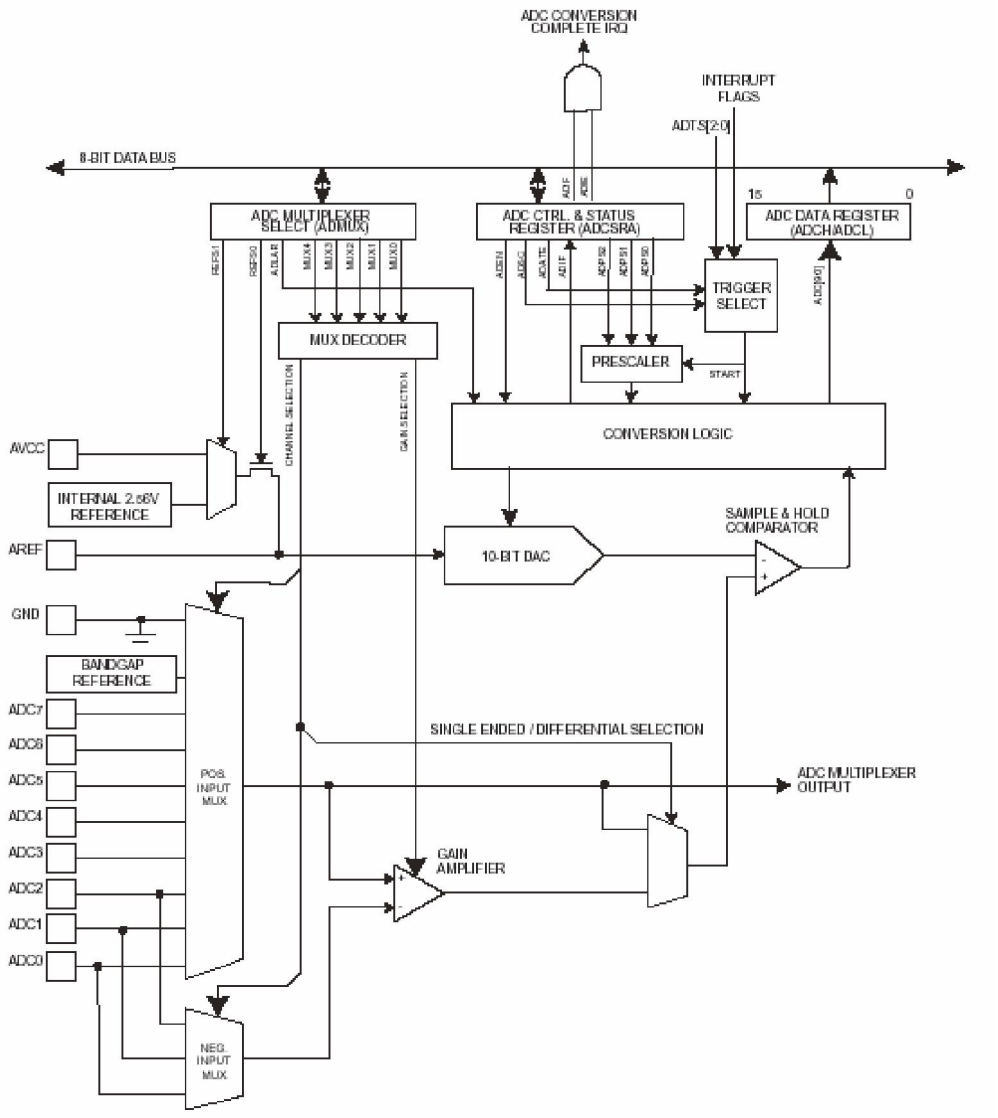


Рис. 6.2. Структурна схема модуля АЦП

Для живлення модулю АЦП у мікроконтролері передбачені виводи: AVCC (напруга живлення) та AGND (аналогова «земля»). Напруга на виводі AVCC не повинна відрізнятися від напруги живлення мікроконтролера більше ніж на ±0,3В, а аналогова «земля» має бути з’єднана з цифровою.

Мікроконтролер має ще вивід AREF для підключення до АЦП зовнішнього джерела опорної напруги. Напруга на цьому виводі повинна знаходитися в діапазоні $0...V_{CC}$.

АЦП може працювати у двох режимах:

- режим одинарного перетворення. У цьому режимі запуск кожного перетворення ініціюється користувачем;
- режим безперервного перетворення. У цьому режимі запуск перетворень виконується безперервно через певні інтервали часу.

7.2.1 Регістри керування аналогово-цифрового перетворювача

Керування модулем АЦП та контроль його стану здійснюється за допомогою регістрів ADCSRA та ADMUX. Результат перетворення розміщується в регістрах ADCH:ADCL.

Режими роботи АЦП встановлюються в основному за допомогою регістру ADCSRA.

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Біти регістру ADCSRA мають наступне призначення («1» - активна):

- ADEN - дозвіл АЦП;
- ADSC - запуск перетворення;
- ADATE - визначення режиму перетворень (одноразові/циклічні);
- ADIF - прапор переривань АЦП;
- ADIE - дозвіл переривань;
- ADPS2, ADPS1, ADPS0 – визначення частоти перетворень.

Перед початком використання АЦП необхідно дозволити його роботу. Для цього необхідно записати «1» в розряд ADEN регістра ADCSRA, а для заборони відповідно «0». Якщо АЦП буде заблокований на час циклу перетворення, то перетворення не буде закінченим (в регістр даних АЦП залишиться результат попереднього перетворення).

Режим роботи АЦП визначається станом розряду ADATE. Якщо він встановлений в «1», АЦП працює в режимі безперервного перетворення. У цьому режимі запуск кожного наступного перетворення здійснюється автоматично після закінчення поточного. Якщо ж біт ADATE скинутий в «0», АЦП працює в режимі одинарного перетворення та запуск кожного перетворення здійснюється за командою користувача.

Запуск перетворення здійснюється установкою в «1» розряду ADSC регістра ADCSRA, а сам цикл перетворення починається за першим зростаючим фронтом тактового сигналу після установки цього розряду. Тривалість циклу складає 13 тактів; вибірка та запам’ятовування вхідного сигналу здійснюється на протязі перших 1,5 тактів. Через 13 тактів

перетворення закінчується, розряд ADSC апаратно скидається в «0» (в режимі одинарного перетворення), та результат перетворення зберігається в регістрі даних АЦП. Одночасно встановлюється прапор переривання ADIF регістра ADCSRA та генерується запит на переривання. Як і прапори останніх переривань, прапор ADIF скидається апаратно при запуску підпрограми обробки переривань від АЦП або програмно – записом в нього логічної «1». Дозвіл переривання здійснюється установкою в «1» розряду ADIE регістра ADCSRA (прапор I регістра SREG також повинен бути встановлений в «1»).

Якщо АЦП працює в режимі безперервного перетворення, новий цикл почнеться одразу ж після запису результату. В режимі одинарного перетворення нове перетворення можна запустити одразу ж після скидання розряду ADSC (до збереження результату поточного перетворення). Однак реально цикл перетворення почнеться не раніше ніж через один такт після закінчення поточного перетворення. При написанні програми потрібно враховувати одну особливість: для першого після включення АЦП перетворення необхідно на 12 тактів більше, ніж для усіх наступних. Це пов'язане з тим, що при запуску першого перетворення спочатку виконується одне «холосте» перетворення, що ініціалізує АЦП. Розряд ADSC в цьому випадку скидається лише після закінчення робочого
перетворення.

Тактовий сигнал модуля АЦП формується подільником, на вхід якого, в свою чергу, надходить тактовий сигнал мікроконтролера. Коефіцієнт ділення цього подільника та, відповідно, тривалість перетворення визначається станом розрядів ADPS2…ADPS0 регістра ADCSRA.

ADPS2	ADPS1	ADPS0	Division Factor
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Найбільша точність перетворення досягається, якщо тактова частота модуля АЦП знаходиться в діапазоні: 50…200 кГц. Відповідно коефіцієнт ділення рекомендується вибирати таким чином, щоб тактова частота модуля АЦП знаходилася у вказаному діапазоні.

Результат перетворення зберігається в регістрі даних АЦП. Оскільки АЦП – 10-розрядний, цей регістр фізично розташований в двох регістрах вводу/виводу ADCH:ADCL, доступних лише для читання. Ці регістри при

підключенні живлення містять значення «$0000». Звертання до цих регістрів (для отримання результату перетворення) повинно виконуватися в певній послідовності: спочатку необхідно прочитати регістр ADCL, а потім ADCH. Ця вимога пов’язана з тим, що після звернення до регістру ADCL процесор блокує доступ до регістрів даних зі сторони АЦП до тих пір, поки не буде прочитаний регістр ADCH. Завдяки цьому можна бути впевненим, що при читанні регістрів в них будуть знаходитися складові одного й того ж результату. Відповідно, якщо чергове перетворення закінчиться до звернення до регістру ADCH, результат перетворення буде втраченим.

Керування вхідним мультиплексором модуля АЦП здійснюється за допомогою регістра ADMUX.

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Розряди MUX4…MUX0 цього регістра визначають номер активного каналу (номер аналогового входу, що підключений до входу АЦП). Стан цих розрядів можна змінити в будь-який час, однак, якщо це буде зроблено під час циклу перетворення, зміна каналу відбудеться лише після завершення перетворення. Завдяки цьому в режимі безперервного перетворення можна легко реалізувати сканування каналів. Під цим терміном в даному випадку розуміють послідовне перетворення сигналів декількох каналів.

Розряди REFS1-REFS0 відповідають за вибір джерела опорної напруги.

REFS1	REFS0	Voltage Reference Selection
0	0	AREF, Internal Vref turned off
0	1	AVCC with external capacitor at AREF pin
1	0	Reserved
1	1	Internal 2.56V Voltage Reference with external capacitor at AREF pin

Розряд ADLAR відповідає за тип вирівнювання результату аналого-цифрового перетворення в регістрі даних АЦП.

ADLAR =0

15	14	13	12	11	10	9	8	
–	–	–	–	–	–	ADC9	ADC8	ADCH
ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL

ADLAR =1

15	14	13	12	11	10	9	8	
ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
ADC1	ADC0	–	–	–	–	–	–	ADCL

Вирівнювання результату «вліво» (ADLAR=0) використовується у випадку, коли необхідно отримати 10-бітний результат. Якщо достатньо 8-бітного результату, то доцільно використати режим «правої» упаковки (ADLAR=1).

Аналогово-цифровий перетворювач мікроконтролера Mega16 дозволяє працювати як з диференційними, так і з однополярними входами. При роботі з диференційними входами є можливість підсилення сигналів у 10 або 200 раз. У приведеній нижче таблиці наведені константи налаштування біт регістру ADMUX для реалізації різних режимів роботи зі входами аналогово-цифрового перетворювача.

Таблиця вибору активного каналу АЦП

MUX4..0	Single Ended Input	Positive Differential Input	Negative Differential Input	Gain
00000	ADC0	N/A		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000	N/A	ADC0	ADC0	10x
01001		ADC1	ADC0	10x
01010(1)		ADC0	ADC0	200x
01011(1)		ADC1	ADC0	200x
01100		ADC2	ADC2	10x
01101		ADC3	ADC2	10x
01110(1)		ADC2	ADC2	200x
01111(1)		ADC3	ADC2	200x
10000		ADC0	ADC1	1x
10001		ADC1	ADC1	1x
10010		ADC2	ADC1	1x
10011		ADC3	ADC1	1x
10100		ADC4	ADC1	1x
10101		ADC5	ADC1	1x
10110		ADC6	ADC1	1x
10111		ADC7	ADC1	1x
11000		ADC0	ADC2	1x
11001		ADC1	ADC2	1x
11010		ADC2	ADC2	1x
11011		ADC3	ADC2	1x
11100		ADC4	ADC2	1x

7.2.2 Приклад використання АЦП

У другому прикладі приведено програму вимірювання напруги, що формується змінним резистором стенду, та передачі результату на LED-індикатор. У цьому прикладі використано програму виводу трирозрядних чисел на LED-індикатор, розглянуту у 5 прикладі третьої лабораторної роботи.

У прикладі проводилася ініціалізація АЦП на наступний режим роботи:

- АЦП включено (ADEN=1);
- режим одиночних перетворень (ADATE=0);
- дозволені переривання від АЦП (ADIE=1);
- частота тактового сигналу АЦП (частота генератора 10МГц) – 156,2 кГц (1/64 – ADPS2= ADPS1=1);
- підключено внутрішнє джерело опорної напруги (REFS1=REFS0=1);
- визначено нормальний порядок упаковки результату перетворення (ADLAR =0, праве вирівнювання);
- підключено вхідний канал з несиметричним входом ADC0.

Для забезпечення цього режиму роботи необхідно провести ініціалізацію порту А та регістрів ADCSRA, ADMUX.

Приклад 2

```
//---------------------------
//ICC-AVR application builder : 12.01.2019 12:55:39
//Target : M16
//Crystal: 10.000Mhz
#include <iom16v.h>
#include <macros.h>
//---------- Символічні визначення ліній портів та константи
#define   ONE       PD7  //розряд одиниць
#define   TEN       PD6  //розряд десятків
#define   HUND      PD5  //розряд сотень
#define   LED_CS    PD3  //строб запису в буферний регістр
//---------- Масив кодів цифр
const unsigned char digits[] = {0b11000000, 0xF9, 0b10100100,
0xB0, 0x99, 0x92, 0x82, 0xF8, 0x80,  0x90};
//---------------------------
//---------- Визначення змінних
unsigned char indicator  = 1;      //змінна поточних розрядів
//1 - індикація одиниць; 2 - десятків; 3 - індикація сотень
unsigned char ones;                //значення одиниць
unsigned char tens;                //значення десятків
unsigned char hunds;               //значення сотень
//---------------------------
//---------- Функція ініціалізації портів
void port_init(void)
{
PORTA = 0;               // початковий стан порту A
```

```
DDRA  = 0;              // налаштування порту A на прийом
PORTC = 0xFF;           // початковий стан всі "1"
DDRC  = 0xFF;           // налаштування порту C на передачу
PORTD = 0b11100000;     // початковий стан ліній порту
DDRD  = 0xFF;           // налаштування порту D на передачу
}
//---------------------------
//---------- Функція ініціалізації таймера
void timer0_init(void)
{
TCCR0 = 0x00;           //зупинка
TCNT0 = 0xF9;           //константа відліку
OCR0  = 0xFA;           //спрацювання таймеру
TCCR0 = 0x05;           //старт таймеру
}
//---------------------------
//---------- Функція BCD - перетворення
void transform(int out)
{
hunds = out / 100;
tens = (out % 100) / 10;
ones = out % 10;
}
//---------------------------
//---------- Функція переривань переповнення таймера T0
#pragma interrupt_handler timer0_ovf_isr:10
void timer0_ovf_isr(void)
{
indicator++;        //модифікація поточного розряду індикації
switch (indicator)
        {
        case 1:                     //індикація одиниць
               {
               PORTC = digits[ones]; //встановити код символу
               PORTD |= (1<<LED_CS); //формування строб
               PORTD &= ~(1<<LED_CS);
               PORTD |= (1<<HUND);   //активізація розряду
               PORTD &=~(1<<ONE);
               }; break;
       case 2:                      //індикація десятків
              {
               PORTC = digits[tens];
               PORTD |= (1<<LED_CS);
               PORTD &= ~(1<<LED_CS);
               PORTD |= (1<<ONE);
               PORTD &=~(1<<TEN);
               }; break;
       case 3:                      //індикація сотень
```

```
              {
               PORTC = digits[hunds];
               PORTD |= (1<<LED_CS);
               PORTD &= ~(1<<LED_CS);
               PORTD |= (1<<TEN);
               PORTD &=~(1<<HUND);
               indicator = 0;
               }; break;
       };
TCNT0 = 0xF9;                        //завантаження таймеру
}
//---------- Функція ініціалізації АЦП
void adc_init(void)
{
ADCSR = 0b11101110;
//біти ADEN, ADSC, ADATE, ADIE, ADPS2, ADPS1 - «1»
ADMUX =(1<<REFS1)|(1<<REFS0);      //біти REFS1,REFS0 - «1»
}
//---------- Функція переривань АЦП
#pragma interrupt_handler adc_isr:15
void adc_isr(void)
{
unsigned int input; // число для виводу на індикатор

input = ADC;
transform (input);  //BCD- перетворення числа
}
//---------- Функція ініціалізації мікроконтролера
void init_devices(void)
{
CLI();
port_init();   //ініціалізація портів

adc_init();    //ініціалізація АЦП
timer0_init(); //ініціалізація таймеру
TIMSK = 0x01;  //дозвіл переривань переповнення таймера T0
SEI();         //загальний дозвіл переривань
}
//---------- Головна функція програми
void main(void)
{
init_devices();     //ініціалізація мікроконтролера

while(1)            // нескінченний цикл
     {};
}
//---------------------------
```

7.3 **Цифрова фільтрація флуктуацій напруги, що виміряна АЦП**

Наявність у вимірюваній напрузі пульсацій та високочастотних завад приводять до того, що результати перетворення АЦП можуть змінюватися. Боротьбу з цим явищем проводять з використанням як схемотехнічних, так і програмних засобів. Високочастотні завади фільтруються за допомогою схемотехнічних засобів – завадоперешкоджаючих фільтрів. Для боротьби з низькочастотними пульсаціями використовують як схемотехнічні засоби (низькочастотні фільтри), так і програмні.

Серед програмних методів низькочастотної фільтрації найбільш часто вживають просте усереднювання та ковзаюче усереднювання.

Просте усереднювання передбачає проведення серії вимірювань. По результатам вимірювань визначається середнє арифметичне. Недоліком методу простого усереднювання є те, що результат можливо визначити лише після проведення повної серії вимірювань. Це приводить до суттєвої затримки у часі.

Ковзаюче усереднювання (MAV – Moving Average) широко вживається для фільтрації як у техніці, так і у фінансах, економіці. Найбільш часто вживають наступні типи MAV фільтрів:

- Просте ковзаюче середнє;
- Зважене ковзаюче середнє;
- Експоненціальне ковзаюче середнє.

Термін "ковзаюче середнє" визначає що масив значень які усереднюються, безперервно рухається у часі.

Ковзаюче усереднювання передбачає проведення двох етапів вимірювання. На попередньому етапі на періоді визначення значення напруги проводиться серія вимірювань. Масив значень запам’ятовується. На другому етапі, при визначенні наступних значень напруги, проводиться не серія, а лише одне вимірювання. Результат цього вимірювання розміщується в масиві замість найстарішого результату. Значення напруги визначається як середнє арифметичне записаних у масив даних.

Такий підхід використовується у MAV фільтрі з простим ковзаючим середнім. У випадках, коли необхідно підсилити вагу результатів останніх або попередніх вимірювань, визначення проводять з ваговими коефіцієнтами складових масиву. Величини вагових коефіцієнтів обирають за різними критеріями. У цих випадках отримують MAV фільтри із зваженим, або експоненціальним ковзаючим середнім.

Використання методів фільтрації із ваговими коефіцієнтами дає змогу поліпшити динамічні характеристики MAV фільтрів, зробити їх більш чутливими на певні зміни у вимірюваній напрузі.

ЛІТЕРАТУРА

1. Евстифеев А.В. Микроконтроллеры AVR семейств Tiny и Mega фирмы Atmel – 2-е изд., стер. М.: Издательский дом «Додека –XX1», 2005. – 560с.

2. Голубцов М.С. Микроконтроллеры AVR: от простого к сложному. М.: СОЛОН-Пресс, 2003. – 288с.

3. Гребнев В.В. Микроконтроллеры семейства AVR фирмы Atmel. М.: ИП РадиоСофт, 2002. – 176с.

4. Баранов В.Н. Применение микроконтроллеров AVR: схемы, алгоритмы, программы. М.: Издательский дом «Додека –XX1», 2004. – 288с.

5. Сташин В.В., Урусов А.В., Мологонцева О.Ф. Проектирование цифровых устройств на однокристальных микроконтроллерах. М.: Энергоатомиздат, 1990